

```

RRRRRRRRRRRRRRR      MMM      MMM      SSSSSSSSSSSSSS
RRRRRRRRRRRRRRR      MMM      MMM      SSSSSSSSSSSSSS
RRRRRRRRRRRRRRR      MMM      MMM      SSSSSSSSSSSSSS
RRR      RRR      MMMMMM      MMMMMM      SSS
RRR      RRR      MMMMMM      MMMMMM      SSS
RRR      RRR      MMMMMM      MMMMMM      SSS
RRR      RRR      MMM      MMM      MMM      SSS
RRR      RRR      MMM      MMM      MMM      SSS
RRR      RRR      MMM      MMM      MMM      SSS
RRRRRRRRRRRRRRR      MMM      MMM      SSSSSSSSSS
RRRRRRRRRRRRRRR      MMM      MMM      SSSSSSSSSS
RRRRRRRRRRRRRRR      MMM      MMM      SSSSSSSSSS
RRR      RRR      MMM      MMM      MMM      SSS
RRR      RRR      MMM      MMM      MMM      SSS
RRR      RRR      MMM      MMM      MMM      SSS
RRR      RRR      MMM      MMM      MMM      SSS
RRR      RRR      MMM      MMM      MMM      SSS
RRR      RRR      MMM      MMM      SSSSSSSSSSSSSS
RRR      RRR      MMM      MMM      SSSSSSSSSSSSSS
RRR      RRR      MMM      MMM      SSSSSSSSSSSSSS

```

53

Syn

NTS

NTS
NTS

NTS

NTS
NTS

N13

NTS

NTS
NTS

NTS

NTS
NTSNTS
NTS

NTS

NTS
NTS

NTS

NTS
NTS

NTS

NTS
NTS

NTS

NTS
NTSNTS
NTS

NTS

NTS

NTA

NTS
NTSNTS
NTS

NTS

NTS
NTS

NTS

10

10

NTC

NT
NT

NTS

NT
NT

NT

PI

10

100

100

10

1

11. *Journal of the American Medical Association*, 277, 1996, 1611-1612.

1

```
RRRRRRRR  MM      MM      333333  DDDDDDDD  EEEEEEEEE  LL      EEEEEEEEE  TTTTTTTTT  EEEEEEEEE
RRRRRRRR  MM      MM      333333  DDDDDDDD  EEEEEEEEE  LL      EEEEEEEEE  TTTTTTTTT  EEEEEEEEE
RR      RR  MMMM  MMMM  33      33  DD      DD  EE      LL      EE      TT      EE
RR      RR  MMMM  MMMM  33      33  DD      DD  EE      LL      EE      TT      EE
RR      RR  MM      MM      33      33  DD      DD  EE      LL      EE      TT      EE
RRRRRRRR  MM      MM      33      33  DD      DD  EEEEEEEE  LL      EEEEEEEE  TT      EEEEEEEE
RRRRRRRR  MM      MM      33      33  DD      DD  EEEEEEEE  LL      EEEEEEEE  TT      EEEEEEEE
RR      RR  MM      MM      33      33  DD      DD  EE      LL      EE      TT      EE
RR      RR  MM      MM      33      33  DD      DD  EE      LL      EE      TT      EE
RR      RR  MM      MM      33      33  DD      DD  EE      LL      EE      TT      EE
RR      RR  MM      MM      33      33  DD      DD  EE      LL      EE      TT      EE
RR      RR  MM      MM      333333  DDDDDDDD  EEEEEEEEE  LLLLLLLLLL  EEEEEEEEE  TT      EEEEEEEEE
RR      RR  MM      MM      333333  DDDDDDDD  EEEEEEEEE  LLLLLLLLLL  EEEEEEEEE  TT      EEEEEEEEE
```

```
LL      IIIIII  SSSSSSSS
LL      IIIIII  SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLLLL  IIIIII  SSSSSSSS
```

.....


```
1 0001 0 MODULE RM3DELETE (LANGUAGE (BLISS32) ,
2 0002 0 IDENT = 'V04-000'
3 0003 0 ) =
4 0004 1 BEGIN
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
10 0010 1 * ALL RIGHTS RESERVED.
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
17 0017 1 * TRANSFERRED.
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
21 0021 1 * CORPORATION.
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
25 0025 1 *
26 0026 1 *
27 0027 1 *****
28 0028 1
29 0029 1 ++
30 0030 1
31 0031 1 FACILITY: RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
32 0032 1
33 0033 1 ABSTRACT:
34 0034 1 This module handles the deletion of index sequential records.
35 0035 1
36 0036 1
37 0037 1
38 0038 1 ENVIRONMENT:
39 0039 1
40 0040 1 VAX/VMS OPERATING SYSTEM
41 0041 1
42 0042 1 --
43 0043 1
44 0044 1
45 0045 1 AUTHOR: Todd M. Katz CREATION DATE: 14-Jul-1982
46 0046 1
47 0047 1
48 0048 1 MODIFIED BY:
49 0049 1
50 0050 1 V03-025 JWT0181 Jim Teague 15-May-1984
51 0051 1 RMSSQUISH moves too many bytes when squishing the
52 0052 1 the data portion out of deleted records.
53 0053 1
54 0054 1 V03-024 DAS0001 David Solomon 25-Mar-1984
55 0055 1 Fix broken branch to RMSRU_JOURNAL3.
56 0056 1
57 0057 1 V03-023 MCN0003 Maria del C. Nasr 04-Apr-1983
```

58	0058	1	Change linkage of RMSNULLKEY to RL\$JSB.
59	0059	1	
60	0060	1	V03-022 TMK0013 Todd M. Katz 26-Mar-1983
61	0061	1	Change the linkage for RMSRU_JOURNAL3 from RL\$RABREG_467 to
62	0062	1	RL\$RABREG_67.
63	0063	1	
64	0064	1	V03-021 MCN0002 Maria del C. Nasr 24-Mar-1983
65	0065	1	More linkages reorganization.
66	0066	1	
67	0067	1	V03-020 RAS0135 Ron Schaefer 17-mar-1983
68	0068	1	Fix spelling of RJR\$_DELET -> RJR\$_DELETE.
69	0069	1	
70	0070	1	V03-019 TMK0012 Todd M. Katz 16-Mar-1983
71	0071	1	Change the linkage for RMSRU_JOURNAL3 from RL\$RABREG_67 to
72	0072	1	RL\$RABREG_467.
73	0073	1	
74	0074	1	V03-018 TMK0011 Todd M. Katz 16-Mar-1983
75	0075	1	Change the symbol RMSR\$_DELET to RJR\$_DELET.
76	0076	1	
77	0077	1	V03-017 MCN0001 Maria del C. Nasr 24-Feb-1983
78	0078	1	Reorganize linkages
79	0079	1	
80	0080	1	V03-016 TMK0010 Todd M. Katz 08-Jan-1983
81	0081	1	Add support for Recovery Unit Journalling and RU ROLLBACK
82	0082	1	Recovery of ISAM files. This support includes:
83	0083	1	
84	0084	1	1. The restructuring of RMSDELETE3B so that the primary data
85	0085	1	record is unpacked and available for RU journalling before
86	0086	1	any part of the file is permanently modified.
87	0087	1	
88	0088	1	2. The RU Journalling of all \$DELETES which occur on RU
89	0089	1	Journalled files within Recovery Units.
90	0090	1	
91	0091	1	3. Modifications to RMSDELETE_RRV, RMSSQUISH_SIDR, and
92	0092	1	RMSDELETE_UDR so that no space is reclaimed when records of
93	0093	1	RU journalled files are \$DELETED within Recovery Units. The
94	0094	1	RRV, primary data record, or SIDR array element is just
95	0095	1	marked RU_DELETE instead.
96	0096	1	
97	0097	1	4. Modifications to RMSDELETE_RRV, RMSSQUISH_SIDR, and
98	0098	1	RMSDELETE_UDR so that RRVs, primary data records and SIDR
99	0099	1	array elements maybe un-deleted during ROLLBACK of
100	0100	1	prematurely terminated or aborted Recovery Units.
101	0101	1	
102	0102	1	5. The addition of a second parameter (SCAN) to RMSSQUISH_SIDR.
103	0103	1	If this parameter is 1 on entry, RMS will scan the entire
104	0104	1	SIDR array looking for non-deleted elements even if no
105	0105	1	duplicates are allowed in the key of reference. If SCAN is 0
106	0106	1	RMS will immediately delete the entire SIDR as was the case
107	0107	1	previously.
108	0108	1	
109	0109	1	V03-015 TMK0009 Todd M. Katz 05-Jan-1983
110	0110	1	The routine RMSDELETE_SIDR no longer calls the routine
111	0111	1	RMSFND_SDR_ARRAY to position to the SIDR element it is to
112	0112	1	delete. It now performs its own positioning.
113	0113	1	
114	0114	1	V03-014 TMK0008 Todd M. Katz 07-Dec-1982

115	0115	1
116	0116	1
117	0117	1
118	0118	1
119	0119	1
120	0120	1
121	0121	1
122	0122	1
123	0123	1
124	0124	1
125	0125	1
126	0126	1
127	0127	1
128	0128	1
129	0129	1
130	0130	1
131	0131	1
132	0132	1
133	0133	1
134	0134	1
135	0135	1
136	0136	1
137	0137	1
138	0138	1
139	0139	1
140	0140	1
141	0141	1
142	0142	1
143	0143	1
144	0144	1
145	0145	1
146	0146	1
147	0147	1
148	0148	1
149	0149	1
150	0150	1
151	0151	1
152	0152	1
153	0153	1
154	0154	1
155	0155	1
156	0156	1
157	0157	1
158	0158	1
159	0159	1
160	0160	1
161	0161	1
162	0162	1
163	0163	1
164	0164	1
165	0165	1
166	0166	1
167	0167	1
168	0168	1
169	0169	1
170	0170	1
171	0171	1

Change the order in which the various parts of a record are deleted during a \$DELETE. First, eliminate the RRV. Next eliminate the user data record. Finally, the alternate keys which are represented in the primary data record are removed. Previously, the SIDs were eliminated before the primary data record, and during this time a lock was kept on the the primary data bucket. This meant that a bucket lock was being held for quite a long time, and that the routine that positioned to a primary data record by means of an alternate index had to be enhanced with a very complex and very large SIDR re-positioning routine, so that the 1.5 SIDR deadlock case would not exist in version 4. Changing the order of events that take place during a \$DELETE allowed a change in the bucket lock strategy which had the dual benefits of eliminating the 1.5 SIDR deadlock case without the expensive SIDR re-positioning code, and reducing the amount of time a lock on the primary data bucket is kept to a minimum - which is an overall ISAM design goal.

This change is not without its cost. The reason why the old strategy was originally implemented, was so that the primary data record would be available for the extraction of the alternate keys so that the corresponding SIDs could be eliminated. Changing the bucket locking strategy such that the primary data record is deleted and the bucket is released before the SIDs are deleted means that the primary data record must be saved in an auxillary record buffer before it is deleted so that it will be available for alternate key extraction. However, this change is not as expensive as it might seem because if the file's prologue version is 3, the primary data record would have to be unpacked into this same record buffer before the keys could be extracted anyway. Thus, it was a simple matter of unpacking either sooner or later. Any additional cost incurred by this new strategy is born solely by prologue 1 and 2 files which previously could extract the alternate keys without moving the primary data record, and now must perform an additional MOV3. However, the benefits derived from this new strategy more than outweigh the cost of this additional MOV3 required in the case of a prologue version which will hopefully fade out of use.

V03-013 TMK0007 Todd M. Katz 06-Dec-1982
The routine RMSSQUISH_SIDR was recovering the space occupied by a SIDR whenever duplicates were allowed and all the elements in the SIDR were deleted even if the SIDR occupied the physically last position in the SIDR bucket. This had the possibility of creating totally empty SIDR buckets, and the encountering of a totally empty SIDR bucket during a positioning for insertion when duplicates are allowed can not always be correctly handled. Thus, a bug existed in the \$DELETE code which had capability of corrupting SIDR indicies.

To fix this code I have decided that the space occupied by the physically last SIDR in the bucket can never be recovered even if all the elements in the array are deleted when duplicates alternate keys are allowed. At best, if the file is a prologue 3 file, and the element is not the first element in the SIDR array, the space occupied by the RRV pointer can be recovered.

: 172
: 173
: 174
: 175
: 176
: 177
: 178
: 179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
: 228

0172 1
0173 1
0174 1
0175 1
0176 1
0177 1
0178 1
0179 1
0180 1
0181 1
0182 1
0183 1
0184 1
0185 1
0186 1
0187 1
0188 1
0189 1
0190 1
0191 1
0192 1
0193 1
0194 1
0195 1
0196 1
0197 1
0198 1
0199 1
0200 1
0201 1
0202 1
0203 1
0204 1
0205 1
0206 1
0207 1
0208 1
0209 1
0210 1
0211 1
0212 1
0213 1
0214 1
0215 1
0216 1
0217 1
0218 1
0219 1
0220 1
0221 1
0222 1
0223 1
0224 1
0225 1
0226 1
0227 1
0228 1

This fix which I have implemented by re-writing the routine RMSSQUISH_SIDR (both to implement the fix and to optimize the existing code) guarantees both that empty SIDR buckets can never be created when duplicate SIDRs are allowed, and that NRP positioning context is maintained.

V03-012 TMK0006 Todd M. Katz 14-Nov-1982

The routine RM\$DELETE_UDR no longer has to return a value. Previously, it was returning a value because the routine that was responsible for reclaiming space occupied by records that were just marked deleted needed to know whether or no an RRV had been created in the place of the reclaimed record. This is no longer the case, as that routine has been modified to no longer require this piece of information.

V03-011 TMK0005 Todd M. Katz 12-Nov-1982

The routine RM\$FND_SDR_ARRAY requires as implicit input the key size of the SIDR it is to position to in IRB\$B_KEYSZ. The routine RM\$DELETE_SIDR was not setting up the IRAB cell with the key size before calling this routine. Therefore, the possibility existed that RM\$FND_SDR_ARRAY would position to the wrong SIDR array, which would then be deleted. This in fact has been seen, during the course of an \$UPDATE when the old SIDRs that have been changed are removed, and this fix corrects this problem.

V03-010 TMK0004 Todd M. Katz 11-Nov-1982

When SIDRs must be deleted and the file is a prologue 3 file, the record must be unpacked so that the alternate keys can be extracted. If RMS positioned by the primary key of reference then it will already have a fully expanded copy of the primary key in keybuffer 1, and it can use this in the unpacking of the record instead of scanning the bucket to re-expand the primary key when primary key compression is enabled. There is one case when it can not use the primary key in keybuffer 1 like this, and that is when the record being deleted is not the same as the current primary data record. This happens when RMS randomly \$FINDs a record since this operation does not update the NRP context. I was not checking for this case and this fix remedies this.

V03-009 TMK0003 Todd M. Katz 06-Oct-1982

When I completely re-wrote this routine (TMK0001), I broke the deletion of prologue 3 fixed length records, in certain cases, because I had assumed that all prologue 3 records included as part of their record overhead a record size field that needs to be updated when the portion of the prologue 3 primary data record occupied by the data is reclaimed. I thought I had fixed this in TMK0002 (although I forgot to mention it in the audit trail), but actually all I did was fix one \$DELETE case and break others that occur more frequently. What I did was to make the assumption that all fixed length prologue 3 records do not include a record size field. This too is incorrect. Actually, if a prologue 3 record with fixed length records has either key or data compression (or both) enabled, then there is a record size field present as part of the record overhead. If both compression types are disabled and the


```
229      record is fixed size then there is no need for a record size
230      field and one is not present. I was not checking any compression
231      bits, but rather, just for a fixed length record format, before
232      deciding whether or there was a record size field to update and
233      this is what caused the problem in TMK0002.
234
235      V03-008 TMK0002      Todd M. Katz      04-Sep-1982
236      Add support for prologue 3 SDRs. This involves changes
237      only to the routine RMSSQUISH_SIDR.
238
239      The field IRB$B_SRCHFLAGS is now a word. Change all references
240      to it.
241
242      V03-007 KBT0162      Keith B. Thompson      21-Aug-1982
243      Reorganize psects
244
245      V03-006 TMK0001      Todd M. Katz      02-Jul-1982
246
247      New version of $DELETE. This module now incorporates all
248      the routines which were formerly in RM3DELSDR.
249
250      *****
251
252      LIBRARY 'RMSLIB:RMS';
253
254      REQUIRE 'RMSSRC:RMSIDXDEF';
255
256      ! Define default PSECTS for code.
257
258      PSECT
259      CODE = RMSRMS3(PSECT_ATTR),
260      PLIT = RMSRMS3(PSECT_ATTR);
261
262      ! Linkages.
263
264      LINKAGE
265      L_ERROR_LINK1,
266      L_JSB,
267      L_JSB01,
268      L_LINK 7 10 11,
269      L_PRESERVE1,
270      L_RABREG,
271      L_RABREG_4567,
272      L_RABREG_567,
273      L_RABREG_67,
274      L_RABREG_7,
275      L_REC_OVRD,
276      L_SIDR_FIRST,
277
278      ! Local Linkage
279
280      RL$DEL_ALL_SIDR = JSB ()
281      : GLOBAL (R_REC_ADDR,R_IDX_DFN,COMMON_RABREG),
282      RL$SQUISH_DATA = JSB ()
283      : GLOBAL (R_REC_ADDR,R_BKT_ADDR,R_IDX_DFN,R_IFAB);
284
285      ! External Routines
```

```

: 286      0350 1 !
: 287      0351 1 EXTERNAL ROUTINE
: 288      0352 1   RMSCLEAN_BDB      : RL$ERROR_LINK1,
: 289      0353 1   RMSCSEARCH_TREE   : RL$RABREG_67,
: 290      0354 1   RMSEXPAND_KEYD     : RL$JSB01,
: 291      0355 1   RMSEXT_ARRAY_RFA   : RL$RABREG_67,
: 292      0356 1   RMSFIND_BY_ID      : RL$RABREG_567,
: 293      0357 1   RMSGET_NEXT_KEY    : RL$LINK_7_10_11,
: 294      0358 1   RMSGETNXT_ARRAY    : RL$RABREG_67,
: 295      0359 1   RMSKEY_DESC        : RL$RABREG_7,
: 296      0360 1   RMSMOVE            : RL$PRESERVE1,
: 297      0361 1   RMSNULLKEY         : RL$JSB,
: 298      0362 1   RMSREC_OVHD        : RL$REC_OVHD,
: 299      0363 1   RMSRECORD_ID       : RL$RABREG_67,
: 300      0364 1   RMSRECORD_KEY      : RL$PRESERVE1,
: 301      0365 1   RMSRECORD_VBN     : RL$PRESERVE1,
: 302      0366 1   RMSRLSBKT         : RL$PRESERVE1,
: 303      0367 1   RMSRU_JOURNAL3     : RL$RABREG_67 ADDRESSING_MODE( LONG_RELATIVE ),
: 304      0368 1   RMSSIDR_END        : RL$RABREG_67,
: 305      0369 1   RMSSIDR_FIRST      : RL$SIDR_FIRST,
: 306      0370 1   RMSUNPACK_REC      : RL$JSBOT,
: 307      0371 1   RMSUPDELCOM       : RL$RABREG_67;
: 308      0372 1
: 309      0373 1 ! Forward Routines
: 310      0374 1 !
: 311      0375 1 FORWARD ROUTINE
: 312      0376 1   RMSDELETE_RRV      : RL$RABREG_4567,
: 313      0377 1   RMSDELETE_SIDR    : RL$RABREG_7,
: 314      0378 1   RMSDELETE_UDR     : RL$RABREG_4567 NOVALUE,
: 315      0379 1   RMSSQUISH_DATA    : RL$SQUISH_DATA NOVALUE,
: 316      0380 1   RMSSQUISH_SIDR    : RL$RABREG_567;
```


RM\$DEL_ALL_SIDR

```
0381 1 %SBTTL 'RM$DEL_ALL_SIDR'
0382 1 ROUTINE RM$DEL_ALL_SIDR (RECORD_SIZE) : RL$DEL_ALL_SIDR NOVALUE =
0383 1
0384 1 !++
0385 1
0386 1 FUNCTIONAL DESCRIPTION:
0387 1
0388 1 The purpose of this routine is to delete every SIDR array element
0389 1 pointing to the the current primary data record. Towards this goal
0390 1 every secondary key represented in the current primary data record
0391 1 is in turn extracted from the current primary data record which has
0392 1 been saved (in an unpacked form if prologue 3) in a record buffer, used
0393 1 to position to the SIDR array element pointing to the current primary
0394 1 data record in the appropriate index, and that array element is
0395 1 deleted. If the current primary data record does not possess one or
0396 1 more secondary keys either because the record is not of sufficient size
0397 1 or the key is null, or if a fast delete is requested and duplicates of
0398 1 one or more secondary keys are allowed, then the deletion of those
0399 1 secondary keys are bypassed.
0400 1
0401 1 CALLING SEQUENCE:
0402 1
0403 1 RM$DEL_ALL_SIDR()
0404 1
0405 1 INPUT PARAMETERS:
0406 1
0407 1 RECORD_SIZE - size of the user data record in IRB$L_RECBUF
0408 1
0409 1 IMPLICIT INPUTS:
0410 1
0411 1 IDX_DFN - index descriptor for the primary key
0412 1
0413 1 IFAB - address of the IFAB
0414 1 IFB$W_KBUFSZ - size of each of the keybuffers
0415 1 IFB$B_PLG_VER - prologue version of the file
0416 1
0417 1 IRAB - address of the IRAB
0418 1 IRB$L_KEYBUF - address of the contiguous keybuffers
0419 1 IRB$L_RECBUF - address of record unpacking buffer
0420 1
0421 1 RAB - address of the RAB
0422 1 RAB$V_FDL - if set, fast-delete requested
0423 1
0424 1 OUTPUT PARAMETERS:
0425 1 NONE
0426 1
0427 1 IMPLICIT OUTPUTS:
0428 1 NONE
0429 1
0430 1 ROUTINE VALUE:
0431 1 NONE
0432 1
0433 1 SIDE EFFECTS:
0434 1
0435 1 AP and REC_ADDR are trashed.
0436 1 Keybuffer 2 contains the key of the last SIDR deleted.
0437 1
```

```

375 0438 1 !--
376 0439 1
377 0440 2 BEGIN
378 0441 2
379 0442 2 BUILTIN
380 0443 2 AP;
381 0444 2
382 0445 2 EXTERNAL REGISTER
383 0446 2 COMMON RAB_STR,
384 0447 2 R_IDX_DFN_STR,
385 0448 2 R_REC_ADDR_STR;
386 0449 2
387 0450 2 LABEL
388 0451 2 BLOCK;
389 0452 2
390 0453 2 ! Delete all of the secondary keys present in the current user data record.
391 0454 2
392 0455 2 WHILE RMSGET_NEXT_KEY()
393 0456 2 DO
394 0457 2
395 0458 2 ! Each secondary key in the file will in turn become the "current"
396 0459 2 ! secondary key for the purpose of deleting its representative in the
397 0460 2 ! current primary data record from the appropriate index.
398 0461 2
399 0462 2 BLOCK: BEGIN
400 0463 2
401 0464 2 ! If a fast-delete is requested, terminate the deletion of the current
402 0465 2 ! secondary key only if this secondary key allows duplicates. If this
403 0466 2 ! secondary key does not allow duplicates, then a fast delete of it can
404 0467 2 ! not be done, since the error caused by a later attempt to insert a
405 0468 2 ! record with a secondary key that is a duplicate of this one would go
406 0469 2 ! undetected.
407 0470 2
408 0471 2 IF .RAB[RAB$V_FDL]
409 0472 2 AND
410 0473 2 .IDX_DFN[IDX$V_DUPKEYS]
411 0474 2 THEN
412 0475 2 LEAVE BLOCK;
413 0476 2
414 0477 2 ! Check that the current primary data record is of a sufficient size to
415 0478 2 ! include the current secondary key. If it is not, terminate the
416 0479 2 ! deletion process for this secondary key.
417 0480 2
418 0481 2 IF .RECORD_SIZE<0, 16> LSSU .IDX_DFN[IDX$W_MINRECSZ]
419 0482 2 THEN
420 0483 2 LEAVE BLOCK;
421 0484 2
422 0485 2 ! In preparation for positioning to the SIDR array element for this
423 0486 2 ! secondary key of the current primary data record, the secondary key
424 0487 2 ! must be extracted into keybuffer 2.
425 0488 2
426 0489 2 REC_ADDR = .IRAB[IRB$L_RECBUF];
427 0490 2
428 0491 2 ! If this secondary key for the current primary data record is null,
429 0492 2 ! there will not be a SIDR array element in this index pointing to the
430 0493 2 ! current primary data record. Therefore, there is no need to continue
431 0494 2 ! with the process of deleting the current secondary key's
```



```

. TITLE      RM3DELETE
. IDENT      \V04-000\

. EXTRN      RMSCLEAN BDB, RMSCSEARCH TREE
. EXTRN      RMSEXPA ND KEYD, RMSEXT ARRY RFA
. EXTRN      RMSFIND BY ID, RMSGET_NEXT_KEY
. EXTRN      RMSGETNXT ARRAY
. EXTRN      RMSKEY DESC, RMSMOVE
. EXTRN      RMSNUL[KEY, RMSREC OVHD
. EXTRN      RMSRECORD_ID, RMSRECORD KEY
. EXTRN      RMSRECORD_VBN, RMSRLSBKT
. EXTRN      RMSRU JOURNAL$, RMSSIDR END
. EXTRN      RMSSIDR FIRST, RMSUNPACR_REC
. EXTRN      RMSUPDEL COM

. PSECT      RMSRMS3.NOWRT, GBL, PIC.2

```

PC	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418	Op419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

RM3DELETE
V04-000

RM\$DEL_ALL_SIDR

H 11
16-Sep-1984 01:42:30
14-Sep-1984 13:01:19

VAX-11 Bliss-32 V4.0-742
[RMS.SRC]RM3DELETE.B32;1

Page 10
(2)

	0000G	30	00020	BSBW	RM\$NULLKEY
5E	04	C0	00023	ADDL2	#4, SP
D9	50	E9	00026	BLBC	R0, 1\$
5C	03	D0	00029	MOVL	#3, AP
50	00B4	CA	3C 0002C	MOVZWL	180(IFAB) R0
	60	B940	9F 00031	PUSHAB	@96(IRAB)[R0]
	0000G	30	00035	BSBW	RM\$RECORD_KEY
5E	04	C0	00038	ADDL2	#4, SP
	0000V	30	0003B	BSBW	RM\$DELETE_SIDR
	C2	11	0003E	BRB	1\$
	10	BA	00040 3\$:	POPR	#^M<R4>
	05	00042		RSB	

0505
0512

0518
0455
0521

; Routine Size: 67 bytes, Routine Base: RM\$RMS3 + 0000


```

: 460      0522 1 %SBTTL 'RMSDELETE3B'
: 461      0523 1 GLOBAL ROUTINE RMSDELETE3B : RLSRABREG =
: 462      0524 1
: 463      0525 1 ++
: 464      0526 1
: 465      0527 1 FUNCTIONAL DESCRIPTION:
: 466      0528 1
: 467      0529 1 This routine directs the deletion of the current primary data record.
: 468      0530 1 To establish a current record, a $GET or $FIND is done. Fast delete
: 469      0531 1 (SIDR entries are not deleted) can only take place when duplicates are
: 470      0532 1 allowed. This is because allowing SIDR entries to not be deleted when
: 471      0533 1 duplicates were not allowed, would mean that the the error condition
: 472      0534 1 "inserting duplicate when not allowed" could not be detected.
: 473      0535 1
: 474      0536 1 The steps involved in deleting the current record are as follows:
: 475      0537 1
: 476      0538 1 1. If the file defines alternate keys or is being RU Journalled, save
: 477      0539 1 the primary data record in a record buffer. If the file is a
: 478      0540 1 prologue 3 file then the primary data record will be saved in
: 479      0541 1 unpacked format.
: 480      0542 1
: 481      0543 1 2. Delete the RRV. The space it occupies maybe completely reclaimed
: 482      0544 1 if the file is a prologue 3 file; otherwise, just the space
: 483      0545 1 occupied by the RRV pointer is recovered.
: 484      0546 1
: 485      0547 1 3. Delete the user data record. This may involve just marking it
: 486      0548 1 deleted, eliminating just the data portion (prologue 3 only), or
: 487      0549 1 eliminating the entire record depending upon the prologue version
: 488      0550 1 of the file, whether duplicate primary keys are allowed, and whether
: 489      0551 1 this primary data record is physically the last record in the primary
: 490      0552 1 data bucket.
: 491      0553 1
: 492      0554 1 4. Delete all secondary keys (unless fast delete is set and duplicates
: 493      0555 1 are allowed). The SIDR will be completely deleted if duplicates
: 494      0556 1 are not allowed, but if duplicates are allowed the SIDR element will
: 495      0557 1 just be marked deleted and the space occupied by the RRV pointer
: 496      0558 1 reclaimed if the file is a prologue 3 file.
: 497      0559 1
: 498      0560 1 NOTE: If this operation is occurring on a RU Journalled file within a
: 499      0561 1 recovery unit then the RRV, primary data record, and all SIDR
: 500      0562 1 elements are marked IRC$V_RU_DELETE and no space is reclaimed.
: 501      0563 1
: 502      0564 1 CALLING SEQUENCE:
: 503      0565 1
: 504      0566 1 RMSDELETE3B()
: 505      0567 1
: 506      0568 1 INPUT PARAMETERS:
: 507      0569 1 NONE
: 508      0570 1
: 509      0571 1 IMPLICIT INPUTS:
: 510      0572 1
: 511      0573 1 IFAB - address of IFAB
: 512      0574 1 IFBSB_NUM_KEYS - number of keys in the file
: 513      0575 1 IFBSB_PLG_VER - prologue version of the file
: 514      0576 1 IFBSV_RUP - if set, Recovery Unit is in progress
: 515      0577 1
: 516      0578 1 IRAB - address of IRAB
```

```
517 0579 1  IRBSB_CUR_KREF - current positioning key of reference
518 0580 1  IRBSW_POS_ID - ID of positioning primary data record
519 0581 1  IRBSL_POS_VBN - VBN of positioning primary data record
520 0582 1  IRBSL_RECBUF - address of record buffer
521 0583 1  IRBSW_UDR_ID - ID of current primary data record
522 0584 1  IRBSL_UDR_VBN - VBN of current primary data record
523 0585 1
524 0586 1 OUTPUT PARAMETERS:
525 0587 1 NONE
526 0588 1
527 0589 1 IMPLICIT OUTPUTS:
528 0590 1
529 0591 1 IRAB
530 0592 1 IRBSV_FIND_LAST - 0, last operation was not a $FIND
531 0593 1 IRBSV_PUTS_LAST - 0, last operation was not a $PUT
532 0594 1 IRBSV_UPDATE - 0, last operation was not an $UPDATE
533 0595 1
534 0596 1 ROUTINE VALUE:
535 0597 1
536 0598 1 CUR - illegal or no current record
537 0599 1 RNL - current record not locked
538 0600 1 SUC - record successfully deleted
539 0601 1 various I/O errors
540 0602 1
541 0603 1 SIDE EFFECTS:
542 0604 1
543 0605 1 If record locking is unnecessary the record locks are not checked for.
544 0606 1 If automatic locking is not specified, the then the deleted record is
545 0607 1 not unlocked.
546 0608 1 If automatic locking is required, then the current primary data record
547 0609 1 is always unlocked, on success or failure.
548 0610 1 If the current process is within a Recovery Unit, and the file is being
549 0611 1 Recovery Unit Journalled, then the operation is RU Journalled
550 0612 1 before any permanent modification to the file takes place
551 0613 1 --
552 0614 1
553 0615 2 BEGIN
554 0616 2
555 0617 2 BUILTIN
556 0618 2 AP;
557 0619 2
558 0620 2 EXTERNAL REGISTER
559 0621 2 COMMON_RAB_STR;
560 0622 2
561 0623 2 GLOBAL REGISTER
562 0624 2 COMMON_IO_STR,
563 0625 2 R_REC_ADDR_STR,
564 0626 2 R_IDX_DFN_STR;
565 0627 2
566 0628 2 LOCAL
567 0629 2 RECORD_SIZE;
568 0630 2
569 0631 2 ! Perform checks common to both $UPDATE and $DELETE such as making sure
570 0632 2 ! there is a current record and that it is locked, and then find the
571 0633 2 ! current record by means of its RFA address. This will access both the
572 0634 2 ! bucket containing the current record and the bucket containing the
573 0635 2 ! current record's RRV, if it has one. The address of the BDB for the
```



```

: 574 0636 2 ! current record bucket will be returned in IRB$$_CURBDB, and the address
: 575 0637 2 ! of the BDB for the RRV bucket will be returned in IRB$$_NXTBDB.
: 576 0638 2
: 577 0639 2 IRAB[IRB$$_UPDATE] = 0;
: 578 0640 2
: 579 0641 2 RETURN_ON_ERROR (RMSUPDELCOM());
: 580 0642 2
: 581 0643 2 ! Retrieve the index descriptor for the primary key.
: 582 0644 2
: 583 0645 2 RMSKEY_DESC (0);
: 584 0646 2
: 585 0647 2 ! If the file contains alternate keys, then save the primary data record
: 586 0648 2 ! (in unpacked format if the file's prologue version is 3), in a record
: 587 0649 2 ! buffer so that the primary data record maybe deleted, and the record will
: 588 0650 2 ! still available. This is so that the alternate keys maybe extracted from
: 589 0651 2 ! it at a later time to be used in the deletion of the corresponding SDRs.
: 590 0652 2
: 591 0653 2 ! If the process is within a recovery unit and the file is being RU
: 592 0654 2 ! Journalled, then unpack the primary data record regardless of whether or
: 593 0655 2 ! not the file defines alternate keys.
: 594 0656 2
: 595 0657 2 IF .IFAB[IFB$$_NUM_KEYS] GTRU 1
: 596 0658 2 OR
: 597 0659 2 .IFAB[IFB$$_RUP]
: 598 0660 2 THEN
: 599 0661 2 BEGIN
: 600 0662 2
: 601 0663 2 LOCAL
: 602 0664 2 REC_SIZE,
: 603 0665 2 SAVE_REC_ADDR : REF BBLOCK;
: 604 0666 2
: 605 0667 2 ! Retrieve the size of the current primary data record, and position
: 606 0668 2 ! past the record overhead to the user data record itself.
: 607 0669 2
: 608 0670 2 SAVE_REC_ADDR = .REC_ADDR;
: 609 0671 2 REC_ADDR = .REC_ADDR + RM$REC_OVHD(0; REC_SIZE);
: 610 0672 2 RECORD_SIZE = .REC_SIZE;
: 611 0673 2
: 612 0674 2 ! If the file is a prologue 3 file, then the current primary data
: 613 0675 2 ! record must be unpacked into the record buffer
: 614 0676 2
: 615 0677 2 IF .IFAB[IFB$$_PLG_VER] GEQU PLG$$_VER_3
: 616 0678 2 THEN
: 617 0679 2 BEGIN
: 618 0680 2
: 619 0681 2 ! If the record is in a special format, then retrieve the true size
: 620 0682 2 ! of the record from the last two bytes in the record's reserved
: 621 0683 2 ! space.
: 622 0684 2
: 623 0685 2 IF .SAVE_REC_ADDR[IRC$$_RU_UPDATE]
: 624 0686 2 THEN
: 625 0687 2 RECORD_SIZE = .(.REC_ADDR + .RECORD_SIZE
: 626 0688 2 - IRC$$_DATSZFLD)<0,16>;
: 627 0689 2
: 628 0690 2 ! As part of the process of unpacking the current primary data
: 629 0691 2 ! record, RMS must extract the primary key from its position in
: 630 0692 2 ! front of the rest of the data record, re-expand it if it is
```

```

: 631      0693  4      ! compressed, and re-integrate it. If the current NRP positioning
: 632      0694  4      ! key of reference is the primary key, then when RMS positioned to
: 633      0695  4      ! the current primary data record it extracted its primary key into
: 634      0696  4      ! keybuffer 1 where it serves as part of the local NRP context. If
: 635      0697  4      ! this is indeed the case, then signal the data record unpacking
: 636      0698  4      ! routine that the primary key for this data record maybe found in
: 637      0699  4      ! keybuffer 1, and that there is no need to again extract and
: 638      0700  4      ! re-expand the primary key as part of the unpacking process;
: 639      0701  4      ! otherwise, signal that the entire unpacking process must be gone
: 640      0702  4      ! through.
: 641      0703  4
: 642      0704  4      ! There is one case when RMS must signal that the entire unpacking
: 643      0705  4      ! process must be gone through even though the primary key is the
: 644      0706  4      ! current key of reference. This is when RMS positioned to the
: 645      0707  4      ! record by means of a random $FIND. This type of operation does
: 646      0708  4      ! not update the NRP context.
: 647      0709  4
: 648      0710  5      IF (.IRAB[IRB$B_CUR_KREF] EQLU 0)
: 649      0711  4          AND
: 650      0712  5          (.IRAB[IRB$W_POS_ID] EQLU .IRAB[IRB$W_UDR_ID])
: 651      0713  4          AND
: 652      0714  5          (.IRAB[IRB$L_POS_VBN] EQLU .IRAB[IRB$L_UDR_VBN])
: 653      0715  4      THEN
: 654      0716  4          AP = 1
: 655      0717  4      ELSE
: 656      0718  4          AP = 0;
: 657      0719  4
: 658      0720  4      RECORD_SIZE = RMSUNPACK_REC (.IRAB[IRB$L_RECBUF], .RECORD_SIZE);
: 659      0721  4
: 660      0722  4      ! If this file is being RU Journalled (Only Prologue 3 files are
: 661      0723  4      ! journalled), and the current process is within a Recovery Unit,
: 662      0724  4      ! then RU Journal the current operation and set the state bit
: 663      0725  4      ! IRB$V_RU_DELETE so that the deletions are done such that no space
: 664      0726  4      ! at all is reclaimed.
: 665      0727  4
: 666      0728  4      IF .IFAB[IFB$V_RUP]
: 667      0729  4      THEN
: 668      0730  5          BEGIN
: 669      0731  5              REC_ADDR = .IRAB[IRB$L_RECBUF];
: 670      0732  5              RETURN_ON_ERROR (RMSRU_JOURNAL3 (RJRS_DELETE,
: 671      0733  5                  .IRAB[IRB$L_UDR_VBN],
: 672      0734  5                  .IRAB[IRB$W_UDR_ID],
: 673      0735  5                  .RECORD_SIZE),
: 674      0736  5                  RMSCLEAN_BDB());
: 675      0737  5              IRAB[IRB$V_RU_DELETE] = 1;
: 676      0738  4          END;
: 677      0739  4      END
: 678      0740  4
: 679      0741  4      ! If the file is a prologue 1 or 2 file, then just move the primary data
: 680      0742  4      ! record into the record buffer.
: 681      0743  4
: 682      0744  3      ELSE
: 683      0745  3          RMSMOVE (.RECORD_SIZE, .REC_ADDR, .IRAB[IRB$L_RECBUF]);
: 684      0746  3
: 685      0747  3      ! Position back to the beginning of the primary data record - to the
: 686      0748  3      ! first byte of the current primary data record's overhead.
: 687      0749  3

```


RM3DELETE
V04-000

RMSDELETE3B

M 11
16-Sep-1984 01:42:30
14-Sep-1984 13:01:19

VAX-11 Bliss-32 V4.0-742
[RMS.SRC]RM3DELETE.B32;1

Page 15
(3)

```

: 688      0750      3      REC_ADDR = .SAVE_REC_ADDR;
: 689      0751      2      END;
: 690      0752      2
: 691      0753      2      ! If the current record is not in its original bucket, process the RRV for
: 692      0754      2      ! the current record. For prologue 3 files this involves deleting the RRV
: 693      0755      2      ! entirely. For all other files, just the space occupied by the RRV pointer
: 694      0756      2      ! to the current record is reclaimed. This means that the current record
: 695      0757      2      ! can no longer be found through its secondary keys or by RFA access.
: 696      0758      2
: 697      0759      2      IF (BDB = .IRAB[IRB$L_NXTBDB]) NEQ 0
: 698      0760      2      THEN
: 699      0761      3          BEGIN
: 700      0762      3              IRAB[IRB$L_NXTBDB] = 0;
: 701      0763      3              RETURN_ON_ERROR (RMSDELETE_RRV(), BEGIN
: 702      0764      3                  IRAB[IRB$V_RU_DELETE] = 0;
: 703      0765      3                  RELEASE (IRAB[IRB$L_CURBDB]);
: 704      0766      3                  END);
: 705      0767      2      END;
: 706      0768      2
: 707      0769      2      ! Delete the current primary data record, mark the bucket dirty and release
: 708      0770      2      ! it. If the current record's key is the high key in the primary data
: 709      0771      2      ! bucket, then the current primary data record is just marked deleted.
: 710      0772      2
: 711      0773      2      BDB = .IRAB[IRB$L_CURBDB];
: 712      0774      2      IRAB[IRB$L_CURBDB] = 0;
: 713      0775      2
: 714      0776      2      RMSDELETE_UDR();
: 715      0777      2
: 716      0778      2      BDB[BDB$V_DRT] = 1;
: 717      0779      2      RETURN_ON_ERROR (RMSRLSBKT(0), IRAB[IRB$V_RU_DELETE] = 0);
: 718      0780      2
: 719      0781      2      ! If the file contains alternate keys, delete all the SISR entries for
: 720      0782      2      ! the current record.
: 721      0783      2
: 722      0784      2      IF .IFAB[IFB$B_NUM_KEYS] GTRU 1
: 723      0785      2      THEN
: 724      0786      2          RMSDEL_ALL_SISR (.RECORD_SIZE);
: 725      0787      2
: 726      0788      2      ! Clear the IRB$V_RU_DELETE state bit regardless of whether this operation
: 727      0789      2      ! was or wasn't R0 Journalled, and then return success.
: 728      0790      2
: 729      0791      2      IRAB[IRB$V_RU_DELETE] = 0;
: 730      0792      2      RETURN RMSSUCT;
: 731      0793      1      END;

```

```

00FC  8F  BB 00000 RMSDELETE3B::
06  A9          08 8A 00004  PUSHR  #^M<R2,R3,R4,R5,R6,R7>
          0000G 30 00008  BICB2  #8, 6(IRAB)
          50  E8 0000B  BSBW  RMSUPDDEL COM
          0101 31 0000E  BLBS  STATUS, 1$
          7E  D4 00011 1$:  BRW  13$
          0000G 30 00013  CLRL  -(SP)
          BSBW  RMSKEY_DESC

```

```

: 0523
: 0639
: 0641
:
: 0645
:

```

	5E		04	C0	00016	ADDL2	#4, SP		
	01	00B2	CA	91	00019	CMPB	178(IFAB), #1		0657
			09	1A	0001E	BGTRU	2\$		
03	00A2	CA	02	E0	00020	BBS	#2, 162(IFAB), 2\$		0659
			008A	31	00026	BRW	9\$		
	53		56	D0	00029	MOVL	REC_ADDR, SAVE_REC_ADDR		0670
			51	D4	0002C	CLRL	R1		0671
			0000G	30	0002E	BSBW	RMSREC_OVHD		
	56		50	C0	00031	ADDL2	R0, REC_ADDR		
	52		51	D0	00034	MOVL	REC_SIZE, RECORD_SIZE		0672
07	03	00B7	CA	91	00037	CMPB	183(IFAB), #3		0677
			65	1F	0003C	BLSSU	7\$		
	63		06	E1	0003E	BBC	#6, (SAVE_REC_ADDR), 3\$		0685
		FE	A246	9F	00042	PUSHAB	-2(RECORD_SIZE)[REC_ADDR]		0687
	52		9E	3C	00046	MOVZWL	@(SP)+, RECORD_SIZE		
		00C3	C9	95	00049	TSTB	195(IRAB)		0710
			17	12	0004D	BNEQ	4\$		
	00BC	C9	00BA	C9	B1	0004F	CMPW	186(IRAB), 188(IRAB)	0712
			0E	12	00056	BNEQ	4\$		
	00B0	C9	00AC	C9	D1	00058	CMPL	172(IRAB), 176(IRAB)	0714
			05	12	0005F	BNEQ	4\$		
	5C		01	D0	00061	MOVL	#1, AP		0716
			02	11	00064	BRB	5\$		
			5C	D4	00066	CLRL	AP		0718
	51		52	D0	00068	MOVL	RECORD_SIZE, R1		0720
	50	68	A9	D0	0006B	MOVL	104(IRAB), R0		
			0000G	30	0006F	BSBW	RMSUNPACK_REC		
	52		50	D0	00072	MOVL	R0, RECORD_SIZE		
35	00A2	CA	02	E1	00075	BBC	#2, 162(IFAB), 8\$		0728
		56	68	A9	D0	0007B	MOVL	104(IRAB), REC_ADDR	0731
			52	DD	0007F	PUSHL	RECORD_SIZE		0736
	7E	00BC	C9	3C	00081	MOVZWL	188(IRAB), -(SP)		
		00B0	C9	DD	00086	PUSHL	176(IRAB)		
			05	DD	0008A	PUSHL	#5		
		00000000G	EF	16	0008C	JSB	RMSRU_JOURNAL3		
	5E		10	C0	00092	ADDL2	#16, SP		
	05		50	E8	00095	BLBS	STATUS, 6\$		
			0000G	30	00098	BSBW	RMSCLEAN_BDB		
			75	11	0009B	BRB	13\$		
	07	A9	20	88	0009D	BISB2	#32, 7(IRAB)		0737
			0D	11	000A1	BRB	8\$		0677
		68	A9	DD	000A3	PUSHL	104(IRAB)		0745
		0044	8F	BB	000A6	PUSHR	#*M<R2,R6>		
			0000G	30	000AA	BSBW	RMSMOVE		
	5E		0C	C0	000AD	ADDL2	#12, SP		
	56		53	D0	000B0	MOVL	SAVE_REC_ADDR, REC_ADDR		0750
	54	3C	A9	D0	000B3	MOVL	60(IRAB), BDB		0759
			24	13	000B7	BEQL	10\$		
		3C	A9	D4	000B9	CLRL	60(IRAB)		0762
			0000V	30	000BC	BSBW	RMSDELETE_RRV		0766
	51		50	D0	000BF	MOVL	R0, STATUS		
	18		51	E8	000C2	BLBS	STATUS, 10\$		
07	A9		20	8A	000C5	BICB2	#32, 7(IRAB)		
	54		20	A9	D0	000C9	MOVL	32(IRAB), BDB	
			20	A9	D4	000CD	CLRL	32(IRAB)	
			7E	D4	000D0	CLRL	-(SP)		
			0000G	30	000D2	BSBW	RMSRLSBKT		

5E	04	C0	000D5	ADDL2	#4, SP	:
50	51	D0	000D8	MOVL	STATUS, R0	:
	35	11	000DB	BRB	13\$:
54	20	A9	D0 000DD	10\$: MOVL	32(IRAB), BDB	: 0773
	20	A9	D4 000E1	CLRL	32(IRAB)	: 0774
		0000V	30 000E4	BSBW	RM\$DELETE UDR	: 0776
0A	A4	02	88 000E7	BISB2	#2, 10(BDB)	: 0778
		7E	D4 000EB	CLRL	-(SP)	: 0779
		0000G	30 000ED	BSBW	RM\$RLSBKT	:
	5E	04	C0 000F0	ADDL2	#4, SP	:
	06	50	E8 000F3	BLBS	STATUS, 11\$:
07	A9	20	8A 000F6	BICB2	#32, 7(IRAB)	:
		16	11 000FA	BRB	13\$:
	01	00B2	CA 91 000FC	11\$: CMPB	178(IFAB), #1	: 0784
			08 1B 00101	BLEQU	12\$:
			52 DD 00103	PUSHL	RECORD_SIZE	: 0786
		FEB5	30 00105	BSBW	RM\$DEL_ALL_SIDR	:
	5E	04	C0 00108	ADDL2	#4, SP	:
07	A9	20	8A 0010B	12\$: BICB2	#32, 7(IRAB)	: 0791
	50	01	D0 0010F	MOVL	#1, R0	: 0792
		00FC	8F BA 00112	13\$: POPR	#^M<R2,R3,R4,R5,R6,R7>	: 0793
			05 00116	RSB		:

; Routine Size: 279 bytes, Routine Base: RM\$RMS3 + 0043


```
0794 1 %SBTTL 'RMSDELETE_RRV'
0795 1 GLOBAL ROUTINE RMSDELETE_RRV : RL$RABREG_4567 =
0796 1
0797 1 ++
0798 1
0799 1 FUNCTIONAL DESCRIPTION:
0800 1
0801 1 Delete the RRV for the current primary data record. If the file is a
0802 1 prologue 3 file the RRV is entirely deleted; otherwise, it is marked
0803 1 deleted and just the space occupied by the pointer is reclaimed.
0804 1
0805 1 If the state bit IRB$V RU DELETE is set, the RRV is just marked
0806 1 RU_DELETE. Likewise, if the state bit IRB$V RU UNDEL is set, then the
0807 1 RU_DELETE bit in the RRV's control byte is cleared.
0808 1
0809 1 CALLING SEQUENCE:
0810 1
0811 1 RMSDELETE_RRV()
0812 1
0813 1 INPUT PARAMETERS:
0814 1 NONE
0815 1
0816 1 IMPLICIT INPUTS:
0817 1
0818 1 BDB - BDB of buffer with RRV bucket in it
0819 1 BDB$L_ADDR - address of buffer
0820 1
0821 1 IFAB - address of IFAB
0822 1 IFB$B_PLG_VER - prologue version of file
0823 1
0824 1 IRAB
0825 1 IRB$V RU_DELETE - if set, mark RU_DELETE and do not reclaim
0826 1 IRB$V RU_UNDEL - if set, un-delete the RRV
0827 1
0828 1 REC_ADDR - address of record whose RRV is to be deleted
0829 1
0830 1 OUTPUT PARAMETERS:
0831 1 NONE
0832 1
0833 1 IMPLICIT OUTPUTS:
0834 1
0835 1 IDX_DFN - index descriptor for the primary key
0836 1
0837 1 ROUTINE VALUE:
0838 1
0839 1 Value of RLSBKT when writing out bucket with RRV deleted
0840 1
0841 1 SIDE EFFECTS:
0842 1
0843 1 AP destroyed.
0844 1 IDX_DFN is set up for the primary key.
0845 1 The freespace offset in the RRV bucket is updated to reflect the
0846 1 amount of space reclaimed.
0847 1
0848 1 --
0849 1
0850 2 BEGIN
```



```

: 790      0851 2
: 791      0852 2
: 792      0853 2      BUILTIN
: 793      0854 2      AP;
: 794      0855 2
: 795      0856 2      EXTERNAL REGISTER
: 796      0857 2      R_BDB_STR,
: 797      0858 2      COMMON_RAB_STR,
: 798      0859 2      R_IDX_DFN_STR,
: 799      0860 2      R_REC_ADDR_STR;
: 800      0861 2
: 801      0862 2      GLOBAL REGISTER
: 802      0863 2      R_BKT_ADDR_STR;
: 803      0864 2
: 804      0865 2      LOCAL
: 805      0866 2      DEL_RRV_SIZE,
: 806      0867 2      LENGTH,
: 807      0868 2      RRV_SIZE,
: 808      0869 2      SAVE_REC_ADDR;
: 809      0870 2      ! Obtain the key descriptor for the primary key of reference.
: 810      0871 2
: 811      0872 2      RMSKEY_DESC(0);
: 812      0873 2
: 813      0874 2      SAVE_REC_ADDR = .REC_ADDR;
: 814      0875 2
: 815      0876 2      ! Extract the RRV ID of the current primary data record.
: 816      0877 2
: 817      0878 2      AP = RMSRECORD_ID();
: 818      0879 2
: 819      0880 2      ! Position to the RRV to be deleted, the RRV for the current primary data
: 820      0881 2      ! record. It is impossible for this positioning to fail as long as the
: 821      0882 2      ! bucket containing the RRV has not been released since RMSFIND_BY_RRV
: 822      0883 2      ! accessed it.
: 823      0884 2
: 824      0885 2      BKT_ADDR = .BDB[BDB$L_ADDR];
: 825      0886 2      RMSFIND_BY_ID();
: 826      0887 2
: 827      0888 2      ! If it is indicated that the RRV should just be marked RU DELETE and that
: 828      0889 2      ! no space should be reclaimed, then do so by setting the RU_DELETE bit
: 829      0890 2      ! within the RRV's control byte.
: 830      0891 2
: 831      0892 2      IF .IRAB[IRB$V_RU_DELETE]
: 832      0893 2      THEN
: 833      0894 2          REC_ADDR[IRC$V_RU_DELETE] = 1
: 834      0895 2
: 835      0896 2      ! If it is indicated that the RRV should be un-deleted, then do so by
: 836      0897 2      ! clearing the RU_DELETE bit in the RRV's control byte.
: 837      0898 2
: 838      0899 2      ELSE
: 839      0900 2          IF .IRAB[IRB$V_RU_UNDEL]
: 840      0901 2          THEN
: 841      0902 2              REC_ADDR[IRC$V_RU_DELETE] = 0
: 842      0903 2
: 843      0904 2              ! Delete the RRV reclaiming as much space as is possible.
: 844      0905 2
: 845      0906 2              ELSE
: 846      0907 2                  BEGIN
```

```
: 847      0908      3
: 848      0909
: 849      0910      3
: 850      0911      3
: 851      0912      3
: 852      0913      3
: 853      0914      4
: 854      0915      4
: 855      0916      4
: 856      0917      4
: 857      0918      3
: 858      0919      4
: 859      0920      4
: 860      0921      4
: 861      0922      3
: 862      0923      3
: 863      0924      3
: 864      0925      3
: 865      0926      3
: 866      0927      4
: 867      0928      3
: 868      0929      3
: 869      0930      3
: 870      0931      3
: 871      0932      3
: 872      0933      3
: 873      0934      3
: 874      0935      3
: 875      0936      3
: 876      0937      3
: 877      0938      3
: 878      0939      3
: 879      0940      3
: 880      0941      3
: 881      0942      3
: 882      0943      3
: 883      0944      3
: 884      0945      3
: 885      0946      3
: 886      0947      3
: 887      0948      3
: 888      0949      3
: 889      0950      2
: 890      0951      2
: 891      0952      2
: 892      0953      2
: 893      0954      2
: 894      0955      2
: 895      0956      2
: 896      0957      2
: 897      0958      2
: 898      0959      2
: 899      0960      1

      ! Setup a series of constants to be used in deleting the RRV. These
      ! constants are prologue dependent.
      IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
      THEN
          BEGIN
              RRV_SIZE      = IRC$C_FIXOVHDSZ;
              DEL_RRV_SIZE = 2;
          END
      ELSE
          BEGIN
              RRV_SIZE      = IRC$C_FIXOVHSZ3;
              DEL_RRV_SIZE = 0;
          END;

      ! Delete/Squish the current primary data record's RRV and fix up
      ! the RRV bucket's freespace.
      LENGTH = (.BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])
              - (.REC_ADDR + .RRV_SIZE);

      IF .LENGTH GTRU 0
      THEN
          RMSMOVE (.LENGTH,
                  .REC_ADDR + .RRV_SIZE,
                  .REC_ADDR + .DEL_RRV_SIZE);

          BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE]
                                      - .RRV_SIZE + .DEL_RRV_SIZE;

      ! If the file is not a prologue 3 file, then the RRV of the current
      ! primary data record was just squished. The RRV pointer was
      ! removed, but the control byte and record ID fields remain. In
      ! this case RMS wants to setup the control byte of the squished RRV
      ! to indicate that it has been deleted, is an RRV, and doesn't
      ! contain a pointer.
      IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
      THEN
          REC_ADDR[IRC$B_CONTROL] = IRC$M_RRV OR IRC$M_DELETED
                                  OR IRC$M_NOPTRS;

      END;

      ! Restore the address of the current primary data record and release the
      ! RRV's bucket after marking it dirty.
      REC_ADDR = .SAVE_REC_ADDR;

      BDB[BDB$V_DRT] = 1;
      RETURN RM$RLSBKT (RL$M_WRT_THRU)

      END;
```


				2C	BB	00000	RMSDELETE_RRV::	
		5E		08	C2	00002	PUSHR	#^M<R2,R3,R5> 0795
				7E	D4	00005	SUBL2	#8, SP
				0000G	30	00007	CLRL	-(SP) 0872
		5E		04	C0	0000A	BSBW	RMSKEY_DESC
		6E		56	D0	0000D	ADDL2	#4, SP
				0000G	30	00010	MOVL	REC_ADDR, SAVE_REC_ADDR 0874
		5C		50	D0	00013	BSBW	RMSRECORD_ID 0878
		55		A4	D0	00016	MOVL	R0, AP
			18	0000G	30	0001A	MOVL	24(BDB), BKT_ADDR 0885
				05	E1	0001D	BSBW	RMSFIND_BY_ID 0886
05	07	A9		20	88	00022	BBC	#5, 7(IRAB), 1\$ 0892
		66		53	11	00025	BISB2	#32, (REC_ADDR) 0894
				06	E1	00027	BRB	6\$
05	07	A9		20	8A	0002C	BBC	#6, 7(IRAB), 2\$ 0900
		66		49	11	0002F	BICB2	#32, (REC_ADDR) 0902
				03	CA	00031	BRB	6\$
		03	00B7	09	1E	00036	CMPB	183(IFAB), #3 0912
				07	D0	00038	BGEQU	3\$
		53		02	D0	0003B	MOVL	#7, RRV_SIZE 0915
		AE	04	06	11	0003F	MOVL	#2, DEL_RRV_SIZE 0916
				09	D0	00041	BRB	4\$ 0912
		53		04	AE	00044	MOVL	#9, RRV_SIZE 0920
				04	A5	00047	CLRL	DEL_RRV_SIZE 0921
51		55		52	C1	0004B	MOVZWL	4(BKT_ADDR), R2 0927
50		56		53	C1	0004F	ADDL3	R2, BKT_ADDR, R1
		51		50	C2	00053	ADDL3	RRV_SIZE, REC_ADDR, R0 0928
				0E	13	00056	SUBL2	R0, LENGTH
				04	BE	00058	BEQL	5\$ 0930
				50	DD	0005C	PUSHAB	@DEL_RRV_SIZE[REC_ADDR] 0934
				51	DD	0005E	PUSHL	R0 0933
				0000G	30	00060	PUSHL	LENGTH 0932
		5E		0C	C0	00063	BSBW	RMSMOVE
		52		53	C3	00066	ADDL2	#12, SP
04	50			04	AE	0006A	SUBL3	RRV_SIZE, R2, R0 0937
	A5	50		03	CA	00070	ADDW3	DEL_RRV_SIZE, R0, 4(BKT_ADDR)
		03	00B7	03	1E	00075	CMPB	183(IFAB), #3 0946
				03	1C	00077	BGEQU	6\$
		66		6E	D0	0007A	MOVB	#28, (REC_ADDR) 0949
		56		02	88	0007D	MOVL	SAVE_REC_ADDR, REC_ADDR 0955
	0A	A4		02	DD	00081	BISB2	#2, TO(BDB) 0957
				0000G	30	00083	PUSHL	#2 0958
		5E		0C	C0	00086	BSBW	RMSRLSBKT
				2C	BA	00089	ADDL2	#12, SP
				05	0008B		POPR	#^M<R2,R3,R5> 0960
							RSB	

; Routine Size: 140 bytes, Routine Base: RMSRMS3 + 015A

```
RM$DELETE_SIDR
: 901 0961 1 %SBTTL 'RM$DELETE_SIDR'
: 902 0962 1 GLOBAL ROUTINE RM$DELETE_SIDR : RLSRABREG_7 =
: 903 0963 1
: 904 0964 1 ++
: 905 0965 1
: 906 0966 1 FUNCTIONAL DESCRIPTION:
: 907 0967 1
: 908 0968 1 This routine's responsibility is to position to the SIDR array element
: 909 0969 1 pointing to the current primary data record for a given key of
: 910 0970 1 reference and delete it. The secondary key in keybuffer 2, and the
: 911 0971 1 RFA address of the current primary data record, found as part of the
: 912 0972 1 local NRP context in the IRAB, are utilized in this positioning.
: 913 0973 1 Deletion of the appropriate SIDR array element consists of one of the
: 914 0974 1 following:
: 915 0975 1
: 916 0976 1 1. Removal of the entire SIDR if duplicates are not allowed.
: 917 0977 1
: 918 0978 1 2. Marking the SIDR array element as deleted and not recovering any
: 919 0979 1 space if duplicates are allowed for this key of reference and the
: 920 0980 1 file is a prologue 1 or 2 file.
: 921 0981 1
: 922 0982 1 3. Marking the SIDR array element as deleted and not recovering any
: 923 0983 1 space if duplicates are allowed for this key of reference, the file
: 924 0984 1 is a prologue 3 file, and the element is the first element in the SIDR
: 925 0985 1 array.
: 926 0986 1
: 927 0987 1 4. Marking the SIDR element deleted and squishing out the space
: 928 0988 1 occupied by the RRV pointer if duplicates are allowed for this key
: 929 0989 1 of reference, the file is a prologue 3 file, and the element is not
: 930 0990 1 the first element in the SIDR array.
: 931 0991 1
: 932 0992 1 5. Removal of the entire SIDR array if duplicates are allowed, this is
: 933 0993 1 the first SIDR with this key value, the SIDR is not the physically
: 934 0994 1 last SIDR in the bucket, and ever single element within the SIDR
: 935 0995 1 array has been deleted.
: 936 0996 1
: 937 0997 1 CALLING SEQUENCE:
: 938 0998 1
: 939 0999 1 RM$DELETE_SIDR()
: 940 1000 1
: 941 1001 1 INPUT PARAMETERS:
: 942 1002 1 NONE
: 943 1003 1
: 944 1004 1 IMPLICIT INPUTS:
: 945 1005 1
: 946 1006 1 IDX_DFN - address of index descriptor
: 947 1007 1 IDX$B_KEYSZ - size of alternate key
: 948 1008 1
: 949 1009 1 IRAB - address of IRAB
: 950 1010 1 IRB$W_UDR_ID - RFA VBN of the current primary data record
: 951 1011 1 IRB$L_UDR_VBN - RFA ID of the current primary data record
: 952 1012 1
: 953 1013 1 OUTPUT PARAMETERS:
: 954 1014 1 NONE
: 955 1015 1
: 956 1016 1 IMPLICIT OUTPUTS:
: 957 1017 1
```



```
RM$DELETE_SIDR
1018 1  IRAB - address of IRAB
1019 1  IRB$B_KEYSZ - size of alternate key for key of reference
1020 1  IRB$B_STOPLEVEL - level of index to search to (set to 0)
1021 1
1022 1  ROUTINE VALUE:
1023 1
1024 1  Status of the RMSRLSBKT call (success or error) that released
1025 1  the modified bucket.
1026 1  BUG - if the SIDR array element could not be located
1027 1
1028 1  SIDE EFFECTS:
1029 1
1030 1  Modified bucket is released.
1031 1  IRB$V_POSDELETE set within IRB$W_SRCHFLGS.
1032 1
1033 1  --
1034 1
1035 2  BEGIN
1036 2
1037 2  EXTERNAL REGISTER
1038 2  COMMON RAB_STR,
1039 2  R_IDX_DFN_STR;
1040 2
1041 2  GLOBAL REGISTER
1042 2  COMMON IO_STR,
1043 2  R_REC_ADDR_STR;
1044 2
1045 2  LABEL
1046 2  FIND_ELEMENT;
1047 2
1048 2  LOCAL
1049 2  BEGIN_OF_SIDR;
1050 2
1051 2  ! Since RMS is going to position so it can delete a SIDR array element,
1052 2  ! set the appropriate search flag, and make sure the key size is set up.
1053 2
1054 2  IRAB[IRB$B_STOPLEVEL] = 0;
1055 2  IRAB[IRB$W_SRCHFLGS] = IRB$M_POSDELETE;
1056 2  IRAB[IRB$B_KEYSZ] = .IDX_DFN[IDX$B_KEYSZ];
1057 2
1058 2  ! Position to the SIDR array element pointing to the current primary data
1059 2  ! record for this key of reference. This loop will only be exited either
1060 2  ! when the array element has been located or all SIDR elements with this
1061 2  ! key value are exhausted.
1062 2
1063 2  FIND_ELEMENT:
1064 2  BEGIN
1065 2
1066 2  LOCAL
1067 2  END_OF_SIDR,
1068 2  ID,
1069 2  STATUS,
1070 2  VBN;
1071 2
1072 2  WHILE 1
1073 2  DO
1074 4  BEGIN
```

```
1015 1075 4
1016 1076 4      ! If RMS is unable to find an array element pointing to the current
1017 1077 4      ! primary data record, then something is very wrong. Return an internal
1018 1078 4      ! bug error, and save the status from RM$CSEARCH_TREE, in the RABs STV
1019 1079 4      ! field.
1020 1080 4
1021 1081 5      IF NOT (STATUS = RM$CSEARCH_TREE())
1022 1082 4      THEN
1023 1083 5          BEGIN
1024 1084 5              RAB[RAB$STV] = .STATUS;
1025 1085 5              RETURN RM$ERR(BUG);
1026 1086 4          END;
1027 1087 4
1028 1088 4      ! Prepare to search the SIDR array for the element pointing to the
1029 1089 4      ! current primary data record.
1030 1090 4
1031 1091 4      BEGIN OF SIDR = .REC_ADDR;
1032 1092 4      END_OF_SIDR = RM$SIDR_END();
1033 1093 4
1034 1094 4      ! Position to the first array element in the SIDR array.
1035 1095 4
1036 1096 4      REC_ADDR = RM$SIDR_FIRST(0);
1037 1097 4
1038 1098 4      ! Search the current SIDR array for the element corresponding to the
1039 1099 4      ! current primary data record.
1040 1100 4
1041 1101 4      WHILE .REC_ADDR LSSA .END_OF_SIDR
1042 1102 4      DO
1043 1103 4
1044 1104 4          ! If after extracting out the RFA pointer from the current SIDR
1045 1105 4          ! array element, RMS finds that it does indeed point to the
1046 1106 4          ! current primary data record, then exit the search loop
1047 1107 4
1048 1108 4          IF RM$EXT_ARRAY_RFA (VBN, ID)
1049 1109 4              AND
1050 1110 5              (.IRAB[IRB$W_UDR_ID] EQLU .ID)
1051 1111 4              AND
1052 1112 5              (.IRAB[IRB$L_UDR_VBN] EQLU .VBN)
1053 1113 4          THEN
1054 1114 4              LEAVE FIND_ELEMENT
1055 1115 4
1056 1116 4          ! If the current array element is deleted or does not point to the
1057 1117 4          ! current primary data record then proceed to the next element in
1058 1118 4          ! the SIDR array.
1059 1119 4
1060 1120 4          ELSE
1061 1121 4              RM$GETNXT_ARRAY();
1062 1122 3          END;
1063 1123 2      END;
1064 1124 2
1065 1125 2      ! Delete the SIDR array pointing to the current primary data record
1066 1126 2      ! for this key of reference. The deletion rules are stated above.
1067 1127 2
1068 1128 2      BDB = .IRAB[IRB$L_CURBDB];
1069 1129 2      IRAB[IRB$L_CURBDB] = 0;
1070 1130 2
1071 1131 2      BKT_ADDR = .BDB[BDB$L_ADDR];
```



```
: 1072      1132  2      RMSSQUISH_SIDR (0, .BEGIN_OF_SIDR);  
: 1073      1133  2  
: 1074      1134  2      ! Mark the bucket dirty, and release it.  
: 1075      1135  2      !  
: 1076      1136  2      BDB[BDB$V_DRT] = 1;  
: 1077      1137  2      RETURN RM$RLSBKT(0);  
: 1078      1138  2  
: 1079      1139  1      END;
```

```
007C  8F  BB  00000  RMSDELETE_SIDR::  
5E      08  C2  00004  PUSH  #^M<R2,R3,R4,R5,R6>      : 0962  
41      A9  94  00007  SUBL2 #8, SP      : 1054  
42  A9      04  B0  0000A  CLRB  65(IRAB)      : 1055  
00A6  C9      A7  90  0000E  MOVW  #4, 66(IRAB)      : 1056  
20      0000G  30  00014  1$:  MOVB  32(IDX_DFN), 166(IRAB)      : 1081  
54      50  D0  00017  BSBW  RM$CSEARCH_TREE  
0B      54  E8  0001A  MOVL  R0, STATUS  
OC  A8      54  D0  0001D  BLBS  STATUS, 2$  
50      8F  3C  00021  MOVL  STATUS, 12(RAB)      : 1084  
8434      5E  11  00026  MOVZWL #33844, R0      : 1085  
53      56  D0  00028  2$:  BRB  6$  
0000G  30  0002B  MOVL  REC_ADDR, BEGIN_OF_SIDR      : 1091  
55      50  D0  0002E  BSBW  RM$SIDR_END      : 1092  
7E  D4  00031  MOVL  R0, END_OF_SIDR  
0000G  30  00033  CLRL  -(SP)      : 1096  
5E      04  C0  00036  BSBW  RM$SIDR_FIRST  
56      50  D0  00039  ADDL2 #4, SP  
55      56  D1  0003C  3$:  MOVL  R0, REC_ADDR  
D3  1E  0003F  CMPL  REC_ADDR, END_OF_SIDR      : 1101  
5E  DD  00041  BGEQU  1$  
08  AE  9F  00043  PUSHL  SP      : 1108  
0000G  30  00046  PUSHAB VBN  
5E      08  C0  00049  BSBW  RM$EXT_ARRAY_RFA  
11      50  E9  0004C  ADDL2 #8, SP  
6E  00BC  C9      00  ED  0004F  BLBC  R0, 4$  
10      08  12  00056  CMPZV #0, #16, 188(IRAB), ID      : 1110  
04  AE  00B0      C9  D1  00058  BNEQ  4$  
05  13  0005E  CMPL  176(IRAB), VBN      : 1112  
0000G  30  00060  4$:  BEQL  5$  
D7  11  00063  BSBW  RM$GETNXT_ARRAY      : 1121  
54      A9  D0  00065  5$:  BRB  3$  
20      A9  D4  00069  MOVL  32(IRAB), BDB      : 1128  
20      A4  D0  0006C  CLRL  32(IRAB)      : 1129  
18      53  DD  00070  MOVL  24(BDB), BKT_ADDR      : 1131  
7E  D4  00072  PUSHL  BEGIN_OF_SIDR      : 1132  
0000V  30  00074  CLRL  -(SP)  
5E      04  C0  00077  BSBW  RMSSQUISH_SIDR  
OA  A4      02  88  0007A  ADDL2 #4, SP  
6E  D4  0007E  BISB2 #2, 10(BDB)      : 1136  
0000G  30  00080  CLRL  (SP)      : 1137  
5E      04  C0  00083  BSBW  RM$RLSBKT  
5E      08  C0  00086  6$:  ADDL2 #4, SP  
ADDL2 #8, SP      : 1139
```

RM3DELETE
V04-000

RMSDELETE_SIDR

K 12
16-Sep-1984 01:42:30
14-Sep-1984 13:01:19

VAX-11 Bliss-32 V4.0-742
[RMS.SRC]RM3DELETE.B32;1

Page 26
(5)

007C 8F BA 00089
05 0008D

POPR #^M<R2,R3,R4,R5,R6>
RSB

;
;

; Routine Size: 142 bytes, Routine Base: RMSRMS3 + 01E6


```
1081 1140 1 %SBTTL 'RMSDELETE_UDR'
1082 1141 1 GLOBAL ROUTINE RMSDELETE_UDR : RLSRABREG_4567 NOVALUE =
1083 1142 1
1084 1143 1 ++
1085 1144 1
1086 1145 1 FUNCTIONAL DESCRIPTION:
1087 1146 1
1088 1147 1 This routine's responsibility is the deletion of a primary data record.
1089 1148 1 Most but not all of the time, the record being deleted is the current
1090 1149 1 primary data record. The rules for how primary data records are deleted
1091 1150 1 are as follows:
1092 1151 1
1093 1152 1 1. If the primary data record is marked deleted, then the entire record
1094 1153 1 is always deleted.
1095 1154 1
1096 1155 1 2. If duplicate primary keys are not allowed, and the record is not the
1097 1156 1 last primary data record in the bucket then the entire primary data
1098 1157 1 record is deleted.
1099 1158 1
1100 1159 1 3. If duplicate primary keys are not allowed, and the record is the
1101 1160 1 last primary data record in the bucket then the primary data record
1102 1161 1 is marked deleted, and the space occupied by the data portion of the
1103 1162 1 record is reclaimed if the file's prologue version is 3.
1104 1163 1
1105 1164 1 4. If duplicate primary keys are allowed then the primary data record
1106 1165 1 is marked deleted, and the space occupied by the data portion of the
1107 1166 1 record is recovered if the file's prologue version is 3.
1108 1167 1
1109 1168 1 5. If the state bit IRB$V RU_DELETE is set, then the primary data
1110 1169 1 record is just marked RU_DELETE and no space is reclaimed.
1111 1170 1
1112 1171 1 6. If the state bit IRB$V RU_UNDEL is set, then the primary data record
1113 1172 1 is un-deleted by clearing the RU_DELETE bit within the record control
1114 1173 1 byte.
1115 1174 1
1116 1175 1 7. If the primary data record is completely deleted, the record was in
1117 1176 1 its original bucket (ie - a RRV does not exist), and the file's
1118 1177 1 prologue version is 1 or 2, then a two-byte RRV is created at the
1119 1178 1 end of the bucket for this record to prevent its ID from being
1120 1179 1 recycled.
1121 1180 1
1122 1181 1 CALLING SEQUENCE:
1123 1182 1
1124 1183 1 RMSDELETE_UDR()
1125 1184 1
1126 1185 1 INPUT PARAMETERS:
1127 1186 1 NONE
1128 1187 1
1129 1188 1 IMPLICIT INPUTS:
1130 1189 1
1131 1190 1 BDB - address of BDB for primary data bucket buffer
1132 1191 1 BDB$ADDR - address of primary data bucket buffer
1133 1192 1 BDB$VBN - VBN of primary data bucket
1134 1193 1
1135 1194 1 IDX_DFN - address of index descriptor for primary key
1136 1195 1 IDX$V_DUPKEYS - if set, duplicate primary keys are allowed
1137 1196 1 IDX$V_KEY_COMPR - if set, primary key compression is enabled
```

```
1138 1197 1 |
1139 1198 1 |
1140 1199 1 | IFAB - address of IFAB
1141 1200 1 | IFBSW_KBUFSZ - size of each keybuffer
1142 1201 1 | IFBSB_PLG_VER - prologue version of the file
1143 1202 1 |
1144 1203 1 | IRAB - address of IRAB
1145 1204 1 | IRBSL_KEYBUF - address of the contiguous keybuffers
1146 1205 1 | IRBSV_RU_DELETE - if set, mark RU_DELETE and do not reclaim
1147 1206 1 | IRBSV_RU_UNDEL - if set, un-delete the RRV
1148 1207 1 | REC_ADDR - address of primary data record to be deleted
1149 1208 1 |
1150 1209 1 | OUTPUT PARAMETERS:
1151 1210 1 | NONE
1152 1211 1 |
1153 1212 1 | IMPLICIT OUTPUTS:
1154 1213 1 | NONE
1155 1214 1 |
1156 1215 1 | ROUTINE VALUE:
1157 1216 1 | NONE
1158 1217 1 |
1159 1218 1 | SIDE EFFECTS:
1160 1219 1 |
1161 1220 1 | AP is trashed.
1162 1221 1 | Keybuffer 5 is trashed (if the primary key of the following primary
1163 1222 1 | data record had to be re-expanded).
1164 1223 1 | The freespace offset in the bucket is updated to reflect the amount
1165 1224 1 | of space reclaimed.
1166 1225 1 | REC_ADDR is unchanged. It either points to the deleted record if the
1167 1226 1 | target primary data record could not be completely removed, or
1168 1227 1 | it points to whatever followed the deleted primary data record
1169 1228 1 | (if anything) if it could.
1170 1229 1 | If this is a prologue 1 or 2 file, and the primary data record which was
1171 1230 1 | deleted is in its original bucket, then a two-byte RRV is created
1172 1231 1 | to replace the deleted primary data record, provided the space
1173 1232 1 | occupied by the record was completely recovered.
1174 1233 1 |
1175 1234 1 | --
1176 1235 1 |
1177 1236 2 | BEGIN
1178 1237 2 |
1179 1238 2 | BUILTIN
1180 1239 2 | AP;
1181 1240 2 |
1182 1241 2 | EXTERNAL REGISTER
1183 1242 2 | R_BDB_STR,
1184 1243 2 | COMMON RAB_STR,
1185 1244 2 | R_IDX_DFN_STR,
1186 1245 2 | R_REC_ADDR_STR;
1187 1246 2 |
1188 1247 2 | GLOBAL REGISTER
1189 1248 2 | R_BKT_ADDR_STR;
1190 1249 2 |
1191 1250 2 | FIELD
1192 1251 2 | DELETE_FLAGS =
1193 1252 2 | SET
1194 1253 2 | BUILD_RRV = [0,0,1,0],
```



```
1195 1254 2          LAST_RECORD      = [0,1,1,0],
1196 1255 2          RE_EXPAND_KEY    = [0,2,1,0],
1197 1256 2          TES;
1198 1257 2
1199 1258 2          LOCAL
1200 1259 2          END_OF_BUCKET      : REF BBLOCK,
1201 1260 2          FLAGS              : BLOCK[1,BYTE]
1202 1261 2          FIELD(DELETE_FLAGS),
1203 1262 2          NEXT_REC_ADDR      : REF BBLOCK,
1204 1263 2          REC_OVHD;
1205 1264 2
1206 1265 2          ! If is is indicated that the primary data record should just be marked
1207 1266 2          ! RU_DELETE and that no space should be reclaimed, then do so by setting
1208 1267 2          ! the RU_DELETE bit within the RRV's control byte.
1209 1268 2
1210 1269 2          IF .IRAB[IRB$V_RU_DELETE]
1211 1270 2          THEN
1212 1271 2              BEGIN
1213 1272 2                  REC_ADDR[IRC$V_RU_DELETE] = 1;
1214 1273 2              RETURN;
1215 1274 2              END
1216 1275 2
1217 1276 2          ! If it is indicated that the primary data record should be un-deleted,
1218 1277 2          ! then do so by clearing the RU_DELETE bit in the pimary data record's
1219 1278 2          ! control byte.
1220 1279 2
1221 1280 2          ELSE
1222 1281 2              IF .IRAB[IRB$V_RU_UNDEL]
1223 1282 2              THEN
1224 1283 2                  BEGIN
1225 1284 2                      REC_ADDR[IRC$V_RU_DELETE] = 0;
1226 1285 2                  RETURN;
1227 1286 2                  END;
1228 1287 2
1229 1288 2          ! Obtain the address of the primary data bucket, and compute the first
1230 1289 2          ! free byte in the data bucket.
1231 1290 2
1232 1291 2          FLAGS = 0;
1233 1292 2          BKT_ADDR = .BDB[BDB$L_ADDR];
1234 1293 2          END_OF_BUCKET = .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE];
1235 1294 2
1236 1295 2          ! Obtain the overhead for ALL records in this primary data bucket, and
1237 1296 2          ! compute the address of the first primary data record which would follow
1238 1297 2          ! the primary data record to be deleted.
1239 1298 2
1240 1299 2          BEGIN
1241 1300 2
1242 1301 2          LOCAL
1243 1302 2          REC_SIZE;
1244 1303 2
1245 1304 2          REC_OVHD = RMSREC_OVHD(0; REC_SIZE);
1246 1305 2          NEXT_REC_ADDR = .REC_ADDR + .REC_OVHD + .REC_SIZE;
1247 1306 2          END;
1248 1307 2
1249 1308 2          ! Determine whether the primary data record to be deleted is the last
1250 1309 2          ! record in the bucket, and set the local state flag accordingly.
1251 1310 2
```

```
: 1252      1311  3      IF (.NEXT_REC_ADDR EQLA .END_OF_BUCKET)
: 1253      1312  2              OR
: 1254      1313  2              .NEXT_REC_ADDR[IRC$V_RRV]
: 1255      1314  2      THEN
: 1256      1315  2          FLAGS[LAST_RECORD] = 1;
: 1257      1316  2
: 1258      1317  2      ! If the target primary data record can not be completely deleted either
: 1259      1318  2      ! because duplicates primary keys are allowed or it is the last record
: 1260      1319  2      ! in the bucket, mark the record deleted, and squish out the data portion
: 1261      1320  2      ! of the primary data record if it is squishable.
: 1262      1321  2
: 1263      1322  2      IF NOT .REC_ADDR[IRC$V_DELETED]
: 1264      1323  2          AND
: 1265      1324  2          (.IDX_DFN[IDX$V_DUPKEYS]
: 1266      1325  3              OR
: 1267      1326  3              .FLAGS[LAST_RECORD])
: 1268      1327  2      THEN
: 1269      1328  2          BEGIN
: 1270      1329  3              RMSSQUISH_DATA();
: 1271      1330  3              REC_ADDR[IRC$V_DELETED] = 1;
: 1272      1331  3              RETURN;
: 1273      1332  3          END
: 1274      1333  3
: 1275      1334  3      ! The primary data record can be completely deleted. It is either marked
: 1276      1335  3      ! deleted (the only reason why RMS would be calling this routine would be
: 1277      1336  3      ! to eliminate it entirely), or duplicates are not allowed and it is not
: 1278      1337  3      ! the last primary data record in the bucket.
: 1279      1338  3
: 1280      1339  2      ELSE
: 1281      1340  2          BEGIN
: 1282      1341  3              LOCAL
: 1283      1342  3              UDR_ID;
: 1284      1343  3
: 1285      1344  3              ! If the file is a prologue 1 or 2 file and the primary data record to
: 1286      1345  3              ! be deleted is in its original bucket (ie - there is no RRV for it),
: 1287      1346  3              ! then a two-byte RRV will have to be created for it at the end of the
: 1288      1347  3              ! bucket inorder to reserve its ID and prevent it from being recycled.
: 1289      1348  3
: 1290      1349  3              AP = 3;
: 1291      1350  3
: 1292      1351  3              IF (.IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3)
: 1293      1352  4                  AND
: 1294      1353  3                  (RM$RECORD_VBN() EQLA .BDB[BDB$L_VBN])
: 1295      1354  4              THEN
: 1296      1355  3                  BEGIN
: 1297      1356  4                      FLAGS[BUILD_RRV] = 1;
: 1298      1357  4                      UDR_ID = .REC_ADDR[IRC$B_ID];
: 1299      1358  4                  END;
: 1300      1359  3
: 1301      1360  3
: 1302      1361  3              ! If primary key compression is enabled, and this primary data record
: 1303      1362  3              ! is not the last record in the file, then the key of the following
: 1304      1363  3              ! record, whose front compression is based on this record, will have
: 1305      1364  3              ! to be re-expanded, after this target primary data record is
: 1306      1365  3              ! completely removed. Set the local state bit accordingly and save the
: 1307      1366  3              ! entire key portion (both control bytes and key) of the target primary
: 1308      1367  3              ! data record in keybuffer 5 to be used in re-expanded the key of the
```



```
: 1309      1368      3      ! primary data record that follows.
: 1310      1369      3
: 1311      1370      3      IF .IDX_DFN[IDX$V_KEY_COMPR]
: 1312      1371      3      AND
: 1313      1372      3      NOT .FLAGS[LAST_RECORD]
: 1314      1373      3      THEN
: 1315      1374      4      BEGIN
: 1316      1375      4      FLAGS[RE_EXPAND_KEY] = 1;
: 1317      1376      4
: 1318      1377      4      RM$MOVE (.(.REC_ADDR + .REC_OVHD)<0,8> + 2,
: 1319      1378      4      .REC_ADDR + .REC_OVHD,
: 1320      1379      4      KEYBUF_ADDR(5));
: 1321      1380      3      END;
: 1322      1381      3
: 1323      1382      3      ! If the primary data record being deleted is not the last entity in
: 1324      1383      3      the bucket, recover the space it occupies by shifting everything
: 1325      1384      3      that follows, and update the freespace offset in the bucket
: 1326      1385      3      accordingly. If the primary data record being deleted is the last
: 1327      1386      3      entity in the primary data bucket the space it occupies maybe
: 1328      1387      3      recovered by just adjusting the freespace offset.
: 1329      1388      3
: 1330      1389      4      IF (.NEXT_REC_ADDR LSSA .END_OF_BUCKET)
: 1331      1390      3      THEN
: 1332      1391      3      RM$MOVE (.END_OF_BUCKET - .NEXT_REC_ADDR,
: 1333      1392      3      .NEXT_REC_ADDR,
: 1334      1393      3      .REC_ADDR);
: 1335      1394      3
: 1336      1395      3      BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE]
: 1337      1396      3      - (.NEXT_REC_ADDR - .REC_ADDR);
: 1338      1397      3
: 1339      1398      3      ! If there is a record following the primary data record just deleted,
: 1340      1399      3      whose primary key is to be re-expanded, re-expand it. The routine
: 1341      1400      3      RM$EXPAND_KEYD will take care of re-adjusting the bucket freespace
: 1342      1401      3      offset.
: 1343      1402      3
: 1344      1403      3      IF .FLAGS[RE_EXPAND_KEY]
: 1345      1404      3      THEN
: 1346      1405      3      RM$EXPAND_KEYD (KEYBUF_ADDR(5), .REC_ADDR + .REC_OVHD);
: 1347      1406      3
: 1348      1407      3      ! If a two-byte RRV must be built for the deleted primary data record,
: 1349      1408      3      then build it at the end of the bucket, and adjust the bucket
: 1350      1409      3      freespace offset to reflect the RRV's size.
: 1351      1410      3
: 1352      1411      3      IF .FLAGS[BUILD_RRV]
: 1353      1412      3      THEN
: 1354      1413      4      BEGIN
: 1355      1414      4      END_OF_BUCKET = .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE];
: 1356      1415      4      END_OF_BUCKET[IRCSB_CONTROL] = IRCSM_DELETED OR IRCSM_NOPTRSZ
: 1357      1416      4      OR IRCSM_RRV;
: 1358      1417      4      END_OF_BUCKET[IRCSB_ID] = .UDR ID;
: 1359      1418      4      BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE] + 2;
: 1360      1419      3      END;
: 1361      1420      3
: 1362      1421      2      END;
: 1363      1422      2
: 1364      1423      1      END;
```


			2C	BB	00000	RMSDELETE_UDR::	
		5E	0C	C2	00002	PUSHR	#^M<R2,R3,R5> : 1141
05	07	A9	05	E1	00005	SUBL2	#12, SP : 1269
		66	20	88	0000A	BBC	#5, 7(IRAB), 1\$: 1272
			48	11	0000D	BISB2	#32, (REC_ADDR) : 1271
05	07	A9	06	E1	0000F	BRB	6\$: 1281
		66	20	8A	00014	BBC	#6, 7(IRAB), 2\$: 1284
			3E	11	00017	BICB2	#32, (REC_ADDR) : 1283
			53	94	00019	BRB	6\$: 1291
		55	18	A4	D0	CLRB	FLAGS : 1292
		52	04	A5	3C	MOVL	24(BDB), BKT_ADDR : 1293
		52		55	C0	MOVZWL	4(BKT_ADDR), END_OF_BUCKET : 1293
				51	D4	ADDL2	BKT_ADDR, END_OF_BUCKET : 1304
				0000G	30	CLRL	R1 : 1304
	04	AE		50	D0	BSBW	RMSREC OVHD : 1305
50		56	04	AE	C1	MOVL	R0, REC OVHD : 1305
6E		50		51	C1	ADDL3	REC OVHD, REC_ADDR, R0 : 1311
		52		6E	D1	ADDL3	REC_SIZE, R0, NEXT_REC_ADDR : 1311
				05	13	CMPL	NEXT_REC_ADDR, END_OF_BUCKET : 1313
03	00	BE		03	E1	BEQL	3\$: 1315
		53		02	88	BBC	#3, @NEXT_REC_ADDR, 4\$: 1322
11		66		02	E0	BISB2	#2, FLAGS : 1324
		04	1C	A7	E8	BBS	#2, (REC_ADDR), 7\$: 1326
09		53		01	E1	BLBS	28(IDX_DFN), 5\$: 1329
				0000V	30	BBC	#1, FLAGS, 7\$: 1330
		66		04	88	BSBW	RMS\$SQUISH_DATA : 1330
				008B	31	BISB2	#4, (REC_ADDR) : 1328
		5C		03	D0	BRW	12\$: 1350
		03	00B7	CA	91	MOVL	#3, AP : 1352
				11	1E	CMPB	183(IFAB), #3 : 1354
				0000G	30	BGEQU	8\$: 1357
	1C	A4		50	D1	BSBW	RMS\$RECORD_VBN : 1358
				08	12	CMPL	R0, 28(BDB) : 1370
		53		01	88	BNEQ	8\$: 1372
	08	AE	01	A6	9A	BISB2	#1, FLAGS : 1375
22	1C	A7		06	E1	MOVZBL	1(REC_ADDR), UDR_ID : 1379
1E		53		01	E0	BBC	#6, 28(IDX_DFN), -9\$: 1378
		53		04	88	BBS	#1, FLAGS, -9\$: 1377
		50	00B4	CA	3C	BISB2	#4, FLAGS : 1379
			60	B940	DF	MOVZWL	180(IFAB), R0 : 1378
			08	BE46	9F	PUSHAL	@96(IRAB)[R0] : 1377
		7E	0C	BE46	9A	PUSHAB	@REC_OVHD[REC_ADDR] : 1377
		6E		02	C0	MOVZBL	@REC_OVHD[REC_ADDR], -(SP) : 1389
				0000G	30	ADDL2	#2, (SP) : 1393
		5E		0C	C0	BSBW	RMS\$MOVE : 1392
		52		6E	D1	ADDL2	#12, SP : 1391
				10	1E	CMPL	NEXT_REC_ADDR, END_OF_BUCKET : 1389
				56	DD	BGEQU	10\$: 1393
			04	AE	DD	PUSHL	REC_ADDR : 1392
7E		52	08	AE	C3	PUSHL	NEXT_REC_ADDR : 1391
				0000G	30	SUBL3	NEXT_REC_ADDR, END_OF_BUCKET, -(SP) : 1391
		5E		0C	C0	BSBW	RMS\$MOVE : 1391
						ADDL2	#12, SP : 1391

RM3DELETE
V04-000

RM\$DELETE_UDR

E 13

16-Sep-1984 01:42:30

14-Sep-1984 13:01:19

VAX-11 Bliss-32 V4.0-742

[RMS.SRC]RM3DELETE.B32;1

Page 33

(6)

50		56	6E	C3	000B1	10\$:	SUBL3	NEXT REC ADDR, REC_ADDR, R0	:	1396	
	04	A5	50	A0	000B5		ADDW2	R0, 4(BKT_ADDR)	:		
12		53	02	E1	000B9		BBC	#2, FLAGS, 11\$:	1403	
51		56	04	AE	C1	000BD	ADDL3	REC_OVHD, REC_ADDR, R1	:	1405	
		50	00B4	CA	3C	000C2	MOVZWL	180(1FAB), R0	:		
		50	60	B940	DE	000C7	MOVAL	@96(IRAB)(R0), R0	:		
				0000G	30	000CC	BSBW	RM\$EXPAND KEYD	:		
		13		53	E9	000CF	11\$:	BLBC	FLAGS, 12\$:	1411
		52	04	A5	3C	000D2	MOVZWL	4(BKT_ADDR), END OF BUCKET	:	1414	
		52		55	C0	000D6	ADDL2	BKT_ADDR, END OF BUCKET	:		
		62		1C	90	000D9	MOVB	#28, (END OF BUCKET)	:	1416	
	01	A2	08	AE	90	000DC	MOVB	UDR_ID, 1(END OF BUCKET)	:	1417	
	04	A5		02	A0	000E1	ADDW2	#2, 4(BKT_ADDR)	:	1418	
		5E		0C	C0	000E5	12\$:	ADDL2	#12, SP	:	1423
				2C	BA	000E8	POPR	#^M<R2,R3,R5>	:		
				05	000EA		RSB		:		

; Routine Size: 235 bytes, Routine Base: RM\$RMS3 + 0274

```
RM3DELETE
V04-000
1366 1424 1 %SBTTL 'RM3SQUISH DATA'
1367 1425 1 ROUTINE RM3SQUISH_DATA : RL3SQUISH_DATA NOVALUE =
1368 1426 1
1369 1427 1 ++
1370 1428 1
1371 1429 1 FUNCTIONAL DESCRIPTION:
1372 1430 1
1373 1431 1 This routine's responsibility is the deletion of the data part of
1374 1432 1 the current primary data record. This deletion can only take place if
1375 1433 1 the file is a prologue 3 file.
1376 1434 1
1377 1435 1 CALLING SEQUENCE:
1378 1436 1
1379 1437 1 RM3SQUISH_DATA()
1380 1438 1
1381 1439 1 INPUT PARAMETERS:
1382 1440 1 NONE
1383 1441 1
1384 1442 1 IMPLICIT INPUTS:
1385 1443 1
1386 1444 1 BKT_ADDR - address of the primary data bucket
1387 1445 1
1388 1446 1 IDX_DFN - address of the primary key index descriptor
1389 1447 1 - if set, key compression is enabled
1390 1448 1 - size of the key
1391 1449 1 - if set, record compression is enabled
1392 1450 1
1393 1451 1 IFAB - address of the IFAB
1394 1452 1 IFB3B_PLG_VER - prologue version of the file
1395 1453 1
1396 1454 1 REC_ADDR - address of the current primary data record
1397 1455 1
1398 1456 1 OUTPUT PARAMETERS:
1399 1457 1 NONE
1400 1458 1
1401 1459 1 IMPLICIT OUTPUTS:
1402 1460 1 NONE
1403 1461 1
1404 1462 1 ROUTINE VALUE:
1405 1463 1 NONE
1406 1464 1
1407 1465 1 SIDE EFFECTS:
1408 1466 1
1409 1467 1 The freespace in the bucket is updated to reflect the space reclaimed.
1410 1468 1
1411 1469 1 --
1412 1470 2 BEGIN
1413 1471 2
1414 1472 2 EXTERNAL REGISTER
1415 1473 2 R_BKT_ADDR_STR,
1416 1474 2 R_IDX_DFN_STR,
1417 1475 2 R_IFAB_STR,
1418 1476 2 R_REC_ADDR_STR;
1419 1477 2
1420 1478 2 GLOBAL REGISTER
1421 1479 2 R_RAB,
1422 1480 2 R_IRAB,
```



```
: 1423      1481 2      R_IMPURE,  
: 1424      1482 2      R_BDB;  
: 1425      1483 2  
: 1426      1484 2      LOCAL  
: 1427      1485 2      REC_SIZE,  
: 1428      1486 2      KEY_SIZE,  
: 1429      1487 2      REC_OVHD,  
: 1430      1488 2      SIZE;  
: 1431      1489 2  
: 1432      1490 2      ! If this is not a prologue 3 file then nothing can be done; however, if  
: 1433      1491 2      ! this is a prologue 3 file then as the primary key is always kept separate  
: 1434      1492 2      ! from the data portion of a prologue 3 data record, the data portion  
: 1435      1493 2      ! of the current primary data record can always be squished out, and its  
: 1436      1494 2      ! space recovered.  
: 1437      1495 2  
: 1438      1496 2      IF .IFAB[IFB$B_PLG_VER] NEQ PLG$C_VER_3  
: 1439      1497 2      THEN  
: 1440      1498 2      RETURN;  
: 1441      1499 2  
: 1442      1500 2      ! Obtain the size of the record overhead and the size of the current  
: 1443      1501 2      ! primary data record. Note that the size of the key (and any key specific  
: 1444      1502 2      ! control bytes) is always included as part of the size of the current  
: 1445      1503 2      ! primary data record.  
: 1446      1504 2  
: 1447      1505 2      REC_OVHD = RM$REC_OVHD(0; REC_SIZE);  
: 1448      1506 2  
: 1449      1507 2      ! Compute the contribution of the primary key of the record to the size of  
: 1450      1508 2      ! the current primary data record. If primary key compression is enabled,  
: 1451      1509 2      ! then the key size will include the two bytes of key compression overhead.  
: 1452      1510 2  
: 1453      1511 2      IF .IDX_DFN[IDX$V_KEY_COMPR]  
: 1454      1512 2      THEN  
: 1455      1513 2      KEY_SIZE = (.REC_ADDR + .REC_OVHD)<0,8> + 2  
: 1456      1514 2      ELSE  
: 1457      1515 2      KEY_SIZE = .IDX_DFN[IDX$B_KEYSZ];  
: 1458      1516 2  
: 1459      1517 2      ! Compute the size of the data portion of the current primary data record.  
: 1460      1518 2      ! If the current primary data record consists of the primary key alone,  
: 1461      1519 2      ! return, as there is no data portion to squish out.  
: 1462      1520 2  
: 1463      1521 3      IF ((SIZE = .REC_SIZE - .KEY_SIZE) EQLU 0)  
: 1464      1522 2      THEN  
: 1465      1523 2      RETURN;  
: 1466      1524 2  
: 1467      1525 2      ! Squish out the data portion of the current primary data record.  
: 1468      1526 2  
: 1469      1527 3      RM$MOVE ((.BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])  
: 1470      1528 2      - (.REC_ADDR + .REC_OVHD + .REC_SIZE),  
: 1471      1529 2      .REC_ADDR + .REC_OVHD + .REC_SIZE,  
: 1472      1530 2      .REC_ADDR + .REC_OVHD + .KEY_SIZE);  
: 1473      1531 2  
: 1474      1532 2      ! Update the record size of the current primary data record to reflect  
: 1475      1533 2      ! the squishing out of the data portion of the record. NOTE that if the  
: 1476      1534 2      ! record is fixed length and both key and record compression are disabled,  
: 1477      1535 2      ! then there will be no record size field to update.  
: 1478      1536 2  
: 1479      1537 3      IF NOT (.IFAB[IFB$B_RFMORG] EQLU FAB$C_FIX
```

```
: 1480      1538      3
: 1481      1539      3
: 1482      1540      3
: 1483      1541      3
: 1484      1542      3
: 1485      1543      3
: 1486      1544      3
: 1487      1545      3
: 1488      1546      3
: 1489      1547      3
: 1490      1548      3
: 1491      1549      3
: 1492      1550      3
: 1493      1551      3
: 1494      1552      1

      AND
      NOT .IDX_DFN[IDX$V_KEY_COMPR]
      AND
      NOT .IDX_DFN[IDX$V_REC_COMPR])
THEN
      (.REC_ADDR + .REC_OVHD - 2)<0,16> = (.REC_ADDR + .REC_OVHD - 2)<0,16>
      - .SIZE;

      ! Update the freespace pointer in the bucket to reflect the space that
      ! has been recovered by the squishing out of the data portion of the
      ! current primary data record.
      BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE] - .SIZE;

END;
```

			0B1C	8F	BB	00000	RMSSQUISH DATA:		
		03	00B7	CA	91	00004	PUSHR	#^M<R2,R3,R4,R8,R9,R11>	: 1425
				58	12	00009	CMPB	183(IFAB), #3	: 1496
				51	D4	0000B	BNEQ	5\$	
				0000G	30	0000D	CLRL	R1	: 1505
				50	D0	00010	BSBW	RM\$REC_OVHD	
		52		06	E1	00013	MOVL	R0, REC_OVHD	
09	1C	A7		6246	9A	00018	BBC	#6, 28(IDX_DFN), 1\$: 1511
		54		02	C0	0001C	MOVZBL	(REC_OVHD)[REC_ADDR], KEY_SIZE	: 1513
		54		04	11	0001F	ADDL2	#2, KEY_SIZE	
				04	11	0001F	BRB	2\$	
		54	20	A7	9A	00021	MOVZBL	32(IDX_DFN), KEY_SIZE	: 1515
53		51		54	C3	00025	SUBL3	KEY_SIZE, REC_SIZE, SIZE	: 1521
				38	13	00029	BEQL	5\$	
50		56		52	C1	0002B	ADDL3	REC_OVHD, REC_ADDR, R0	: 1530
				6440	9F	0002F	PUSHAB	(KEY_SIZE)[R0]	
		51		50	C0	00032	ADDL2	R0, R1	: 1529
				51	DD	00035	PUSHL	R1	
		50	04	A5	3C	00037	MOVZWL	4(BKT_ADDR), R0	: 1527
		50		55	C0	0003B	ADDL2	BKT_ADDR, R0	
7E		50		51	C3	0003E	SUBL3	R1, R0, -(SP)	: 1528
				0000G	30	00042	BSBW	RM\$MOVE	
		5E		0C	C0	00045	ADDL2	#12, SP	
		01	50	AA	91	00048	CMPB	80(IFAB), #1	: 1537
				0A	12	0004C	BNEQ	3\$	
05	1C	A7		06	E0	0004E	BBS	#6, 28(IDX_DFN), 3\$: 1539
			1C	A7	95	00053	TSTB	28(IDX_DFN)	: 1541
				07	18	00056	BGEQ	4\$	
			FE	A246	9F	00058	PUSHAB	-2(REC_OVHD)[REC_ADDR]	: 1544
		9E		53	A2	0005C	SUBW2	SIZE, 3(SP)+	
	04	A5		53	A2	0005F	SUBW2	SIZE, 4(BKT_ADDR)	: 1550
			0B1C	8F	BA	00063	POPR	#^M<R2,R3,R4,R8,R9,R11>	: 1552
				05	00067		RSB		

; Routine Size: 104 bytes, Routine Base: RM\$RMS3 + 035F


```
: 1496      1553 1 %SBTTL 'RMSSQUISH_SIDR'
: 1497      1554 1 GLOBAL ROUTINE RMSSQUISH_SIDR (SCAN, BEGIN_OF_SIDR) : RL$RABREG_567 =
: 1498      1555 1
: 1499      1556 1 ++
: 1500      1557 1
: 1501      1558 1 FUNCTIONAL DESCRIPTION:
: 1502      1559 1
: 1503      1560 1     This routine's responsibility is to delete the SIDR array element
: 1504      1561 1     pointing to the current primary data record for this key of reference.
: 1505      1562 1     Deletion of the SIDR array element goes according to one of the
: 1506      1563 1     following rules:
: 1507      1564 1
: 1508      1565 1     1. Removal of the entire SIDR if duplicates are not allowed. NOTE that
: 1509      1566 1     if the input parameter SCAN is 1 and the file is a prologue 3 file
: 1510      1567 1     then for the purpose of this SIDR deletion it is assumed that this
: 1511      1568 1     key of reference does allow duplicates (See rules 2 through 5).
: 1512      1569 1
: 1513      1570 1     2. Marking the SIDR array element as deleted and not recovering any
: 1514      1571 1     space if duplicates are allowed for this key of reference and the
: 1515      1572 1     file is a prologue 1 or 2 file.
: 1516      1573 1
: 1517      1574 1     3. Marking the SIDR array element as deleted and not recovering any
: 1518      1575 1     space if duplicates are allowed for this key of reference, the file
: 1519      1576 1     is a prologue 3 file, and the element is the first element in the
: 1520      1577 1     SIDR array.
: 1521      1578 1
: 1522      1579 1     4. Marking the SIDR element deleted and squishing out the space
: 1523      1580 1     occupied by the RRV pointer if duplicates are allowed for this key
: 1524      1581 1     of reference, the file is a prologue 3 file, and the element is not
: 1525      1582 1     the first element in the SIDR array.
: 1526      1583 1
: 1527      1584 1     5. Removal of the entire SIDR array if duplicates are allowed, this is
: 1528      1585 1     the first SIDR with this key value, the SIDR is not the physically
: 1529      1586 1     last SIDR in the bucket, and every single element within the SIDR
: 1530      1587 1     array has been deleted.
: 1531      1588 1
: 1532      1589 1     6. If the state bit IRB$V_RU_DELETE is set, then the SIDR array element
: 1533      1590 1     is just marked RU_DELETE and no space is reclaimed.
: 1534      1591 1
: 1535      1592 1     7. If the state bit IRB$V_RU_UNDEL is set, then the SIDR array element
: 1536      1593 1     is un-deleted by clearing the RU_DELETE bit within the element's
: 1537      1594 1     control byte.
: 1538      1595 1
: 1539      1596 1 CALLING SEQUENCE:
: 1540      1597 1
: 1541      1598 1     BSBW RMSSQUISH_SIDR()
: 1542      1599 1
: 1543      1600 1 INPUT PARAMETERS:
: 1544      1601 1
: 1545      1602 1     SCAN          - if 1, scan the current SIDR array (if Prologue 3 file)
: 1546      1603 1
: 1547      1604 1     BEGIN_OF_SIDR - pointer to the beginning of the SIDR record
: 1548      1605 1
: 1549      1606 1 IMPLICIT INPUTS:
: 1550      1607 1
: 1551      1608 1     BKT_ADDR      - address of the SIDR bucket
: 1552      1609 1
```

```

: 1553      1610 1 |      IDX_DFN      - address of the index descriptor
: 1554      1611 1 |      -IDXSV_DUPKEYS - if set, duplicate keys are allowed
: 1555      1612 1 |      -IDXSV_KEY_COMPR - if set, SIDR key compression is enabled
: 1556      1613 1 |
: 1557      1614 1 |      IFAB      - address of IFAB
: 1558      1615 1 |      IFBSW_KBUFSZ - size of one of the contiguous keybuffers
: 1559      1616 1 |      IFBSB_PLG_VER - prologue version of file
: 1560      1617 1 |
: 1561      1618 1 |      IRAB      - address of IRAB
: 1562      1619 1 |      IRBSL_KEYBUF - address of the contiguous keybuffers
: 1563      1620 1 |      IRBSV_RU_DELETE - if set, mark RU_DELETE and do not reclaim
: 1564      1621 1 |      IRBSV_RU_UNDEL - if set, un-delete the RRV
: 1565      1622 1 |
: 1566      1623 1 |      REC_ADDR      - address of the SIDR array element
: 1567      1624 1 |
: 1568      1625 1 |      OUTPUT PARAMETERS:
: 1569      1626 1 |      NONE
: 1570      1627 1 |
: 1571      1628 1 |      IMPLICIT OUTPUTS:
: 1572      1629 1 |
: 1573      1630 1 |      REC_ADDR      - address of next SIDR if the entire SIDR was deleted
: 1574      1631 1 |      otherwise unchanged.
: 1575      1632 1 |
: 1576      1633 1 |      ROUTINE VALUE:
: 1577      1634 1 |
: 1578      1635 1 |      1      - some space was recovered.
: 1579      1636 1 |      0      - no space was recovered.
: 1580      1637 1 |
: 1581      1638 1 |      SIDE EFFECTS:
: 1582      1639 1 |
: 1583      1640 1 |      Keybuffer 5 will have been trashed, if any key re-expansion occurred.
: 1584      1641 1 |      The freespace in the bucket is updated to reflect the space reclaimed.
: 1585      1642 1 |      If the SIDR is completely deleted, SIDR key compression is enabled, and
: 1586      1643 1 |      a SIDR follows the completely deleted SIDR, then the key of this
: 1587      1644 1 |      following SIDR will have been re-expanded.
: 1588      1645 1 |
: 1589      1646 1 |      --
: 1590      1647 1 |
: 1591      1648 2 |      BEGIN
: 1592      1649 2 |
: 1593      1650 2 |      EXTERNAL REGISTER
: 1594      1651 2 |      R_BKT_ADDR_STR,
: 1595      1652 2 |      COMMON_RAB_STR,
: 1596      1653 2 |      R_IDX_DFN_STR,
: 1597      1654 2 |      R_REC_ADDR_STR;
: 1598      1655 2 |
: 1599      1656 2 |      LABEL
: 1600      1657 2 |      DUPS;
: 1601      1658 2 |
: 1602      1659 2 |      LOCAL
: 1603      1660 2 |      DELETE_START,
: 1604      1661 2 |      DELETE_END,
: 1605      1662 2 |      FLAGS      : BLOCK[1],
: 1606      1663 2 |      LENGTH,
: 1607      1664 2 |      NEXT_REC_ADDR,
: 1608      1665 2 |      RECORD_OVHD,
: 1609      1666 2 |      SAVE_REC_ADDR : REF BBLOCK;
```



```
: 1610      1667 2
: 1611      1668
: 1612      1669
: 1613      1670
: 1614      1671
: 1615      1672
: 1616      1673
: 1617      1674
: 1618      1675
: 1619      1676
: 1620      1677
: 1621      1678
: 1622      1679
: 1623      1680
: 1624      1681
: 1625      1682
: 1626      1683
: 1627      1684
: 1628      1685
: 1629      1686
: 1630      1687
: 1631      1688
: 1632      1689
: 1633      1690
: 1634      1691
: 1635      1692
: 1636      1693
: 1637      1694
: 1638      1695
: 1639      1696
: 1640      1697
: 1641      1698
: 1642      1699
: 1643      1700
: 1644      1701
: 1645      1702
: 1646      1703
: 1647      1704
: 1648      1705
: 1649      1706
: 1650      1707
: 1651      1708
: 1652      1709
: 1653      1710
: 1654      1711
: 1655      1712
: 1656      1713
: 1657      1714
: 1658      1715
: 1659      1716
: 1660      1717
: 1661      1718
: 1662      1719
: 1663      1720
: 1664      1721
: 1665      1722
: 1666      1723 3

MAP
    BEGIN_OF_SIDR    : REF BBLOCK;

MACRO
    DELETE_SIDR      = 0,0,1,0 %,
    SQUISH_SIDR      = 0,1,1,0 %,
    RE_EXPAND_KEY    = 0,2,1,0 %;

! If it is indicated that the SIDR array element should just be marked
! RU_DELETE and that no space should be reclaimed, then do so by setting
! the RU_DELETE bit within the element's control byte.
IF .IRAB[IRB$V_RU_DELETE]
THEN
    BEGIN
    REC_ADDR[IRC$V_RU_DELETE] = 1;
    RETURN 0;
    END

! If it is indicated that the SIDR array element should be un-deleted,
! then do so by clearing the RU_DELETE bit in the element's control byte.
ELSE
    IF .IRAB[IRB$V_RU_UNDEL]
    THEN
        BEGIN
        REC_ADDR[IRC$V_RU_DELETE] = 0;
        RETURN 0;
        END;

! Save the address of the current SIDR element, and zero out the local
! flag field.
FLAGS = 0;
SAVE_REC_ADDR = .REC_ADDR;

! Determine the address of the first byte past the end of the current
! SIDR.
BEGIN
LOCAL
    REC_SIZE;

REC_ADDR = .BEGIN OF_SIDR;
RECORD_OVHD = RM$REC_OVHD(-1, REC_SIZE);
NEXT_REC_ADDR = .REC_ADDR + .RECORD_OVHD + .REC_SIZE;
END;

! If this secondary key of reference does not allow duplicate key values
! and either the file's prologue version is 1 or 2; or, the input parameter
! SCAN is 0, then the entire SIDR maybe deleted.
IF NOT .IDX_DFN[IDX$V_DUPKEYS]
AND
(NOT .SCAN
```

```
: 1667      1724      3      OR
: 1668      1725      3      .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3)
: 1669      1726      2      THEN
: 1670      1727      2      FLAGS[DELETE_SIDR] = 1
: 1671      1728      2
: 1672      1729      2      ! If this key of reference does allow duplicate SIDR keys or duplicates are
: 1673      1730      2      ! not allowed but the file's prologue version is 3 and a scan of then entire
: 1674      1731      2      ! SIDR array has been requested (SCAN is set to 1), then mark the current
: 1675      1732      2      ! element as deleted and under certain circumstances, reclaim the space
: 1676      1733      2      ! occupied by the SIDR array element's RRV pointer. Under very restricted
: 1677      1734      2      ! circumstances it will also be possible to reclaim the space occupied by
: 1678      1735      2      ! the entire SIDR.
: 1679      1736      2
: 1680      1737      2      ELSE
: 1681      1738      2      DUPS:
: 1682      1739      3      BEGIN
: 1683      1740      3      SAVE_REC_ADDR[IRC$V_DELETED] = 1;
: 1684      1741      3
: 1685      1742      3      ! If the file is a prologue 2 file then marking the element deleted is
: 1686      1743      3      ! all that can be done.
: 1687      1744      3
: 1688      1745      4      IF (.IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3)
: 1689      1746      3      THEN
: 1690      1747      4      BEGIN
: 1691      1748      4      REC_ADDR = .SAVE_REC_ADDR;
: 1692      1749      4      RETURN 0;
: 1693      1750      4      END
: 1694      1751      4
: 1695      1752      4      ! The file is a prologue 3 file. If every single array element in this
: 1696      1753      4      ! SIDR array is deleted, if the SIDR is not physically the last SIDR in
: 1697      1754      4      ! the bucket (this restriction applies to duplicates keys allowed only)
: 1698      1755      4      ! and if this SIDR is the first such SIDR with this key value in the
: 1699      1756      4      ! file then it will be possible to delete the entire SIDR; otherwise,
: 1700      1757      4      ! the space occupied by the element's RRV pointer is reclaimed unless
: 1701      1758      4      ! it is the first element in the array in which case nothing more can
: 1702      1759      4      ! be done.
: 1703      1760      4
: 1704      1761      3      ELSE
: 1705      1762      4      BEGIN
: 1706      1763      4
: 1707      1764      4      LABEL
: 1708      1765      4      ENTIRE_SIDR;
: 1709      1766      4
: 1710      1767      4      LOCAL
: 1711      1768      4      FIRST_SIDR      : REF BBLOCK;
: 1712      1769      4
: 1713      1770      4      ! Obtain the address of the first array element in the SIDR array.
: 1714      1771      4
: 1715      1772      4      FIRST_SIDR = RM$SIDR_FIRST(0);
: 1716      1773      4
: 1717      1774      4      ! If the first element in the array (which maybe the element being
: 1718      1775      4      ! deleted) is marked deleted, and this SIDR is the first such
: 1719      1776      4      ! record in the file with this key value, then it still maybe
: 1720      1777      4      ! possible to delete the entire SIDR.
: 1721      1778      4
: 1722      1779      4      IF .FIRST_SIDR[IRC$V_DELETED]
: 1723      1780      4      AND
```



```
: 1724      1781  4      .FIRST_SIDR[IRC$V_FIRST_KEY]
: 1725      1782  4      THEN
: 1726      1783  5  ENTIRE_SIDR: BEGIN
: 1727      1784  5
: 1728      1785  5      LOCAL
: 1729      1786  5          SCAN_START;
: 1730      1787  5
: 1731      1788  5      ! If the current SIDR is physically the last SIDR in the bucket
: 1732      1789  5      ! and duplicates keys are allowed then it will not be possible
: 1733      1790  5      ! to reclaim the space occupied by the entire SIDR even if all
: 1734      1791  5      ! its elements are deleted.
: 1735      1792  5
: 1736      1793  6      IF .NEXT_REC_ADDR GEQA (.BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])
: 1737      1794  5          AND
: 1738      1795  5          .IDX_DFN[IDX$V_DUPKEYS]
: 1739      1796  5      THEN
: 1740      1797  5          LEAVE ENTIRE_SIDR;
: 1741      1798  5
: 1742      1799  5      ! Scan the SIDR array starting with the second element up to
: 1743      1800  5      ! but not including the target element making sure that all
: 1744      1801  5      ! these elements have been deleted. If a live element is found
: 1745      1802  5      ! then the space occupied by the entire SIDR can not be
: 1746      1803  5      ! reclaimed.
: 1747      1804  5
: 1748      1805  5      SCAN_START = .FIRST_SIDR + .FIRST_SIDR[IRC$V_PTRSZ]
: 1749      1806  5          + IRC$C_DATPTRBS3
: 1750      1807  5          + 1;
: 1751      1808  5
: 1752      1809  6      IF (.SCAN_START LSSA .SAVE_REC_ADDR)
: 1753      1810  5      THEN
: 1754      1811  5          IF NOT CH$FAIL (CH$FIND NOT CH
: 1755      1812  5              (.SAVE_REC_ADDR - .SCAN_START,
: 1756      1813  5              .SCAN_START,
: 1757      1814  5              %CHAR(IRC$M_DELETED)
: 1758      1815  5              OR
: 1759      1816  5              %CHAR(IRC$M_NOPTRSZ)))
: 1760      1817  5      THEN
: 1761      1818  5          LEAVE ENTIRE_SIDR;
: 1762      1819  5
: 1763      1820  5      ! Scan the SIDR array starting with the first element past the
: 1764      1821  5      ! target element and ending with the last element in the SIDR
: 1765      1822  5      ! making sure that all these elements have been deleted. If a
: 1766      1823  5      ! live element is found then the space occupied by the entire
: 1767      1824  5      ! SIDR can not be reclaimed.
: 1768      1825  5
: 1769      1826  5      SCAN_START = .SAVE_REC_ADDR + .SAVE_REC_ADDR[IRC$V_PTRSZ]
: 1770      1827  5          + IRC$C_DATPTRBS3
: 1771      1828  5          + 1;
: 1772      1829  5
: 1773      1830  6      IF (.SCAN_START LSSA .NEXT_REC_ADDR)
: 1774      1831  5      THEN
: 1775      1832  5          IF NOT CH$FAIL (CH$FIND NOT CH
: 1776      1833  5              (.NEXT_REC_ADDR - .SCAN_START,
: 1777      1834  5              .SCAN_START,
: 1778      1835  5              %CHAR(IRC$M_DELETED)
: 1779      1836  5              OR
: 1780      1837  5              %CHAR(IRC$M_NOPTRSZ)))
```

```
: 1781      1838  5          THEN
: 1782      1839  5          LEAVE ENTIRE_SIDR;
: 1783      1840  5
: 1784      1841  5          ! Every single element in the current SIDR has been found to be
: 1785      1842  5          ! deleted, so the space occupied by the entire SIDR maybe
: 1786      1843  5          ! reclaimed.
: 1787      1844  5          FLAGS[DELETE_SIDR] = 1;
: 1788      1845  5          LEAVE DUPS;
: 1789      1846  5          END;
: 1790      1847  4
: 1791      1848  4
: 1792      1849  4          ! If it is not possible to delete the entire SIDR then set up to
: 1793      1850  4          ! reclaim the space occupied by the element's RRV pointer unless the
: 1794      1851  4          ! element is the first element in the array in which case nothing
: 1795      1852  4          ! more can be done.
: 1796      1853  4          REC_ADDR = .SAVE_REC_ADDR;
: 1797      1854  4
: 1798      1855  4          IF (.REC_ADDR EQLA .FIRST_SIDR)
: 1799      1856  5          THEN
: 1800      1857  4              RETURN 0
: 1801      1858  4          ELSE
: 1802      1859  4              FLAGS[SQUISH_SIDR] = 1;
: 1803      1860  4          END;
: 1804      1861  3          END;
: 1805      1862  2
: 1806      1863  2          ! If the space occupies by the entire SIDR is to be reclaimed, set up to
: 1807      1864  2          ! recover it.
: 1808      1865  2
: 1809      1866  2          IF .FLAGS[DELETE_SIDR]
: 1810      1867  2          THEN
: 1811      1868  2              BEGIN
: 1812      1869  3              DELETE_START = .BEGIN_OF_SIDR;
: 1813      1870  3              DELETE_END = .NEXT_REC_ADDR;
: 1814      1871  3
: 1815      1872  3              ! If key compression is enabled, and this SIDR is not the last SIDR
: 1816      1873  3              ! in the bucket, save the key of the current SIDR in keybuffer 5,
: 1817      1874  3              ! so that it maybe used in expanding the key of the following
: 1818      1875  3              ! record.
: 1819      1876  3              IF .IDX_DFN[IDX$V_KEY_COMPR]
: 1820      1877  3              THEN
: 1821      1878  3                  BEGIN
: 1822      1879  3                      GLOBAL REGISTER
: 1823      1880  4                      R_BDB;
: 1824      1881  4
: 1825      1882  4                      FLAGS[RE_EXPAND_KEY] = 1;
: 1826      1883  4
: 1827      1884  4                      RMSMOVE (.(.REC_ADDR + .RECORD_OVHD)<0,8> + 2,
: 1828      1885  4                      .REC_ADDR + .RECORD_OVHD,
: 1829      1886  4                      KEYBUF_ADDR(5));
: 1830      1887  4
: 1831      1888  4                  END;
: 1832      1889  4              END;
: 1833      1890  3          END;
: 1834      1891  3
: 1835      1892  3          ! If the space occupies by the RRV pointer is to be reclaimed, set up to
: 1836      1893  3          ! recover it.
: 1837      1894  3
```



```
: 1838      1895      3      !
: 1839      1896      3      !
: 1840      1897      3      ELSE
: 1841      1898      3      BEGIN
: 1842      1899      3      DELETE_START = .REC_ADDR + 1;
: 1843      1900      3      DELETE_END   = .DELETE_START + .REC_ADDR[IRC$V_PTRSZ]
: 1844      1901      3      + IRC$C_DATPTRBS3;
: 1845      1902      3
: 1846      1903      3      REC_ADDR[IRC$V_NOPTRSZ] = 1;
: 1847      1904      3      REC_ADDR[IRC$V_PTRSZ]   = 0;
: 1848      1905      3
: 1849      1906      3      ! Update the SIDR size field. As it is currently written, this
: 1850      1907      3      ! updating assumes that the size field is the first two bytes
: 1851      1908      3      ! (and the only two bytes) of the record overhead field.
: 1852      1909      3
: 1853      1910      3      (.BEGIN_OF_SIDR)<0,16> = .(.BEGIN_OF_SIDR)<0,16>
: 1854      1911      3      - (.DELETE_END - .DELETE_START);
: 1855      1912      2      END;
: 1856      1913      2
: 1857      1914      2      ! Recover the space that can be recovered, and update the freespace offset
: 1858      1915      2      ! in the SIDR bucket. If the SIDR is being completely deleted, and it is the
: 1859      1916      2      ! last SIDR in the bucket then there will be nothing to move and only the
: 1860      1917      2      ! bucket's freespace offset need be updated.
: 1861      1918      2
: 1862      1919      2      LENGTH = .BKT_ADDR[BKT$W_FREESPACE] - (.DELETE_END - .BKT_ADDR);
: 1863      1920      2
: 1864      1921      2      IF .LENGTH GTRU 0
: 1865      1922      2      THEN
: 1866      1923      2      BEGIN
: 1867      1924      2
: 1868      1925      2      GLOBAL REGISTER
: 1869      1926      2      R_BDB;
: 1870      1927      2
: 1871      1928      2      RM$MOVE (.LENGTH, .DELETE_END, .DELETE_START);
: 1872      1929      2      END;
: 1873      1930      2
: 1874      1931      2      BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE]
: 1875      1932      2      - (.DELETE_END - .DELETE_START);
: 1876      1933      2
: 1877      1934      2      ! If key compression is enabled, the space occupied by the current SIDR was
: 1878      1935      2      ! completely reclaimed, and a SIDR follows whose key needs to be
: 1879      1936      2      ! re-expanded, do so at this point.
: 1880      1937      2
: 1881      1938      2      IF .FLAGS[RE_EXPAND_KEY]
: 1882      1939      2      THEN
: 1883      1940      2      RM$EXPAND_KEYD (KEYBUF_ADDR(5), .REC_ADDR + .RECORD_OVHD);
: 1884      1941      2
: 1885      1942      2      ! Return indicating that some space has been recovered.
: 1886      1943      2
: 1887      1944      2      RETURN 1;
: 1888      1945      1      END;
```


		5E	0C	C2	00002	PUSHR	#^M<R2,R3,R4>	1554
05	07	A9	05	E1	00005	SUBL2	#12, SP	1680
		66	20	88	0000A	BBC	#5, 7(IRAB), 1\$	1683
			43	11	0000D	BISB2	#32, (REC_ADDR)	1684
05	07	A9	06	E1	0000F	BRB	4\$	1691
		66	20	8A	00014	BBC	#6, 7(IRAB), 2\$	1694
			39	11	00017	BICB2	#32, (REC_ADDR)	1695
			04	AE	D4	BRB	4\$	1701
		54	56	D0	0001C	CLRL	FLAGS	1702
		56	20	AE	D0	MOVL	REC_ADDR, SAVE_REC_ADDR	1702
		51	01	CE	00023	MOVL	BEGIN_OF_SIDR, REC_ADDR	1712
			0000G	30	00026	MNEGL	#1, RT	1713
	08	AE	50	D0	00029	BSBW	RM\$REC_OVHD	
50		56	08	AE	C1	MOVL	R0, RECORD_OVHD	1714
53		50	51	C1	00032	ADDL3	RECORD_OVHD, REC_ADDR, R0	
		0B	1C	A7	E8	ADDL3	REC_SIZE, R0, NEXT_REC_ADDR	
		7E	1C	AE	E9	BLBS	28(IDX_DFN), 3\$	1721
		03	00B7	CA	91	BLBC	SCAN, T0\$	1723
				77	1F	CMPB	183(IFAB), #3	1725
		64	04	88	00045	BLSSU	10\$	
		03	00B7	CA	91	BISB2	#4, (SAVE_REC_ADDR)	1740
				06	1E	CMPB	183(IFAB), #3	1745
		56	54	D0	0004F	BGEQU	5\$	
			0100	31	00052	MOVL	SAVE_REC_ADDR, REC_ADDR	1748
			7E	D4	00055	BRW	17\$	1749
			0000G	30	00057	CLRL	-(SP)	1772
		5E	04	C0	0005A	BSBW	RM\$SIDR_FIRST	
		6E	50	D0	0005D	ADDL2	#4, SP	
5D	00	BE	02	E1	00060	MOVL	R0, FIRST_SIDR	
			00	BE	95	BBC	#2, @FIRST_SIDR, 11\$	1779
			58	18	00068	TSTB	@FIRST_SIDR	1781
		50	04	A5	3C	BGEQ	11\$	
		50	55	C0	0006E	MOVZWL	4(BKT_ADDR), R0	1793
		50	53	D1	00071	ADDL2	BKT_ADDR, R0	
			04	1F	00074	CPL	NEXT_REC_ADDR, R0	
50	00	BE	1C	A7	E8	BLSSU	6\$	1795
		02	00	EF	0007A	BLBS	28(IDX_DFN), 11\$	1805
		51	06	D0	00080	EXTZV	#0, #2, @FIRST_SIDR, R0	1807
		52	05	A041	9E	MOVL	FIRST_SIDR, R1	
		54	52	D1	00088	MOVAB	5(R0)[R1], SCAN_START	
			10	1E	0008B	CPL	SCAN_START, SAVE_REC_ADDR	1809
		54	52	C3	0008D	BGEQU	8\$	
		62	14	3B	00091	SUBL3	SCAN_START, SAVE_REC_ADDR, R0	1812
			02	12	00095	SKPC	#20, -R0, (SCAN_START)	1815
			51	D4	00097	BNEQ	7\$	
			51	D5	00099	CLRL	R1	
			25	12	0009B	TSTL	R1	1816
		02	00	EF	0009D	BNEQ	11\$	
50	64	52	05	A044	9E	EXTZV	#0, #2, (SAVE_REC_ADDR), R0	1826
		53	52	D1	000A7	MOVAB	5(R0)[SAVE_REC_ADDR], SCAN_START	1828
			10	1E	000AA	CPL	SCAN_START, NEXT_REC_ADDR	1830
		53	52	C3	000AC	BGEQU	10\$	
		62	14	3B	000B0	SUBL3	SCAN_START, NEXT_REC_ADDR, R0	1833
			02	12	000B4	SKPC	#20, -R0, (SCAN_START)	1836
			51	D4	000B6	BNEQ	9\$	
			51	D5	000B8	CLRL	R1	
						TSTL	R1	1837

			06	12	000BA		BNEQ	11\$		
	04	AE	01	88	000BC	10\$:	BISB2	#1, FLAGS		1845
			0C	11	000C0		BRB	12\$		1846
		56	54	D0	000C2	11\$:	MOVL	SAVE_REC_ADDR, REC_ADDR		1854
		6E	56	D1	000C5		CMPL	REC_ADDR, FIRST_SIDR		1856
			88	13	000C8		BEQL	4\$		
	04	AE	02	88	000CA		BISB2	#2, FLAGS		1860
		2D	04	AE	E9 000CE	12\$:	BLBC	FLAGS, 13\$		1867
		51	20	AE	D0 000D2		MOVL	BEGIN_OF_SIDR, DELETE_START		1870
		52		53	D0 000D6		MOVL	NEXT_REC_ADDR, DELETE_END		1871
3D		A7		06	E1 000D9		BBC	#6, 28(IDX_DFN), 14\$		1878
	1C	AE		04	88 000DE		BISB2	#4, FLAGS		1885
	04	50	00B4	CA	3C 000E2		MOVZWL	180(IFAB), R0		1889
			60	B940	DF 000E7		PUSHAL	@96(IRAB)[R0]		
			0C	BE46	9F 000EB		PUSHAB	@RECORD_OVHD[REC_ADDR]		1888
		7E	10	BE46	9A 000EF		MOVZBL	@RECORD_OVHD[REC_ADDR], -(SP)		1887
		6E		02	C0 000F4		ADDL2	#2, (SP)		
				0000G	30 000F7		BSBW	RM\$MOVE		
		5E		0C	C0 000FA		ADDL2	#12, SP		
				1C	11 000FD		BRB	14\$		1867
		51	01	A6	9E 000FF	13\$:	MOVAB	1(R6), DELETE_START		1899
50		02		00	EF 00103		EXTZV	#0, #2, (REC_ADDR), R0		1900
	66	52	04	A041	9E 00108		MOVAB	4(R0)[DELETE_START], DELETE_END		1901
		66		10	88 0010D		BISB2	#16, (REC_ADDR)		1903
		66		03	8A 00110		BICB2	#3, (REC_ADDR)		1904
	50	51		52	C3 00113		SUBL3	DELETE_END, DELETE_START, R0		1911
		BE	20	50	A0 00117		ADDW2	R0, @BEGIN_OF_SIDR		
	50	55		52	C3 0011B	14\$:	SUBL3	DELETE_END, BKT_ADDR, R0		1919
		53	04	A5	3C 0011F		MOVZWL	4(BKT_ADDR), R3		
		50		53	C0 00123		ADDL2	R3, LENGTH		
				0A	13 00126		BEQL	15\$		1921
				51	DD 00128		PUSHL	DELETE_START		1928
				05	BB 0012A		PUSHR	#^M<R0,R2>		
			0000G	30	0012C		BSBW	RM\$MOVE		
		5E		0C	C0 0012F		ADDL2	#12, SP		
		51		52	C2 00132	15\$:	SUBL2	DELETE_END, R1		1932
		A5	04	51	A0 00135		ADDW2	R1, 4(BKT_ADDR)		
	12	AE		02	E1 00139		BBC	#2, FLAGS, 16\$		1938
	51	56	08	AE	C1 0013E		ADDL3	RECORD_OVHD, REC_ADDR, R1		1940
		50	00B4	CA	3C 00143		MOVZWL	180(IFAB), R0		
		50	60	B940	DE 00148		MOVAL	@96(IRAB)[R0], R0		
			0000G	30	0014D		BSBW	RM\$EXPAND_KEYD		
		50		01	D0 00150	16\$:	MOVL	#1, R0		1944
				02	11 00153		BRB	18\$		
				50	D4 00155	17\$:	CLRL	R0		1945
		5E		0C	C0 00157	18\$:	ADDL2	#12, SP		
				1C	BA 0015A		POPR	#^M<R2,R3,R4>		
				05	0015C		RSB			

; Routine Size: 349 bytes, Routine Base: RM\$RMS3 + 03C7

:	1889	1946	1	
:	1890	1947	1	END
:	1891	1948	1	
:	1892	1949	0	ELUDOM

PSECT SUMMARY

```
:
:      Name          Bytes          Attributes
: RM$RMS3           1316 NOVEC,NOWRT, RD , EXE,NOSHR, GBL, REL, CON, PIC,ALIGN(2)
```

Library Statistics

```
:
:      File          ----- Symbols ----- Pages Processing
:                   Total   Loaded   Percent   Mapped   Time
: _$255$DUA28:[RMS.OBJ]RMS.L32;1      3109      92      2      154      00:00.4
```

COMMAND QUALIFIERS

```
:
: BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:RM3DELETE/OBJ=OBJ$:RM3DELETE MSRC$:RM3DELETE/UPDATE=(ENH$:RM3DELETE)
: 1893          1950 0
: Size:         1316 code + 0 data bytes
: Run Time:      00:33.6
: Elapsed Time:  01:00.1
: Lines/CPU Min: 3483
: Lexemes/CPU-Min: 15181
: Memory Used:  163 pages
: Compilation Complete
```


0324

AH-BT13A-SE
 VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY